TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Janne Lautamäki

On the Development of Real-Time Multi-User Web Applications

Julkaisu 1167 • Publication 1167

Tampere 2013

Tampereen teknillinen yliopisto. Julkaisu 1167
Tampere University of Technology. Publication 1167

Janne Lautamäki

On the Development of Real-Time Multi-User Web Applications

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB216, at Tampere University of Technology, on the 8th of November 2013, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2013

ISBN 978-952-15-3170-5 (printed)
ISBN 978-952-15-3185-9 (PDF)
ISSN 1459-2045

ABSTRACT

With the increasing popularity of the World Wide Web (WWW), end-user applications are moving from desktop to the browser. Web applications have several benefits over native applications: web applications have worldwide availability for any browsing capable device without prior installations. In addition, web applications are easy to distribute and update – once deployed, a web application is instantly available worldwide and further modifications to the system are propagated automatically. The current trend seems to be that web applications are offering collaboration, social connections, and user to user interactions as key features. This can be seen, for example, in the popularity of Facebook, Flickr, and Twitter.

Despite all the benefits of the Web, web applications are suffering from the shortcomings in underlying technologies. The Web is strongly rooted in information sharing, and the current technical solutions isolate users rather than make them aware of each other. Since the data cannot be pushed from server to a client, the client must always initiate the communication, which causes a considerable impediment for real-time multi-user web applications, like online chats that have several concurrent users continuously interacting with each other. For such systems, it would be practical if the server could push messages to clients. As a further obstacle, most web application frameworks isolate users in their private sessions that only interact indirectly via the database.

The main contribution of this thesis is to make the development of real-time multi-user web applications easier. We elaborate on the difficulties in implementation and design and introduce methods of circumventing them. The main argument is that the Web, the available technology stack, and the frameworks are difficult to use for developing real-time multi-user web applications. However, by selecting the proper approach, the problems can be solved.

In this thesis, we have divided the frameworks in groups based on how they make separation of concerns between the client and the server. The separation is important as it determines the thickness of the client and thus where to locate the business logic and the application state. In addition, it has effect on the synchronization of the state between the clients. To collect experiences and for backing up our assumptions, we have implemented real-time multi-user web applications for several frameworks and studied how the frameworks should be used for enabling real-time multi-user application development.

Keywords: Real-time, Collaboration, Multi-user, Web applications, Thin client, Thick client

ACKNOWLEDGEMENTS

With much gratitude, I wish to acknowledge the following people for their professional support and guidance.

First of all, I would like to thank Professor Tommi Mikkonen, my supervisor, who has always been ready to help me and give me helpful suggestions for my research and this thesis. I would also like to thank Dr. Riku Suomela for giving the first push for this work during the MoMUPE project.

I also wish to express my gratitude for my colleagues (past and present) and all the co-authors. Especially I would like to thank Timo Aho, Janne Kuuskeri, Tommi Mikkonen, Kari Systa, and Mikko Tiusanen who have helped me in proofreading and have all given valuable feedback. I am grateful to my pre-inspectors Joao Araujo, and Jari Porras for gentle criticism and careful work. Furthermore, I thank for my opponents Olaf Droegehorn and Jari Porras beforehand and hope that they will offer a decent but gentle show during the defense.

No acknowledgement would be complete without a word of gratitude to my friends, relatives, grandparents, parents, and my own family for their support and understanding. So, thank you for all that you have done.

Janne Lautamäki
Tampere, October, 8th, 2013

CONTENTS

LIST OF INCLUDED PUBLICATIONS

This thesis is a compound of the following Publications which are referred to in the text by their roman numerals.

I. Janne Lautamäki, Anssi Heiska, Tommi Mikkonen, and Riku Suomela. "Supporting mobile online multiuser service development." In the 3rd IET International Conference on Intelligent Environments. IE 07, pp. 198-204, 2007.

II. Anssi Jääskeläinen and Janne Lautamäki. "Analyzing context-aware service development under MUPE platform." In the Eighth International Workshop on Applications and Services in Wireless Networks. ASWN'08, pp. 26-34. IEEE, 2008.

III. Janne Lautamäki and Riku Suomela. "Using player proximity in mobile multiplayer games: experiences from Sandman." In Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, pp. 248-251. ACM, 2008.

IV. Janne Kuuskeri, Janne Lautamäki, and Tommi Mikkonen. "Peer-to-peer collaboration in the lively kernel." In Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 812-817. ACM, 2010.

V. Timo Aho, Adnan Ashraf, Marc Englund, Joni Katajamäki, Johannes Koskinen, Janne Lautamäki, Antti Nieminen, Ivan Porres, and Ilkka Turunen. "Designing IDE as a Service." Communications of Cloud Software 1, no. 1, p. 10, 2011.

VI. Janne Lautamäki, Antti Nieminen, Johannes Koskinen, Timo Aho, Tommi Mikkonen, and Marc Englund. "CoRED: browser-based Collaborative Real-time Editor for Java web applications." In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, pp. 1307-1316. ACM, 2012.

VII. Juha-Matti Vanhatupa and Janne Lautamäki. "Content Generation in a Collaborative Browser-Based Game Environment." Handbook of Digital Games, ISBN: 978-1-118-32803-3, p. 26, Wiley-IEEE Press, 2014.

1. INTRODUCTION

As the users of web applications are linked via servers, it seems reasonable to assume that they could interact with each other. However, traditionally this has not been the case. Web 2.0, the second phase in the Web‘s evolution [O’Reilly07], offers more interactions and collaboration. It emphasizes social interactions and collective intelligence, and thus presents new opportunities. Along with Web 2.0, the division between the publisher and consumer has faded. This can be seen in social networking sites and wikis, but also in the blogs, the product reviews, and the comment sections. Technically, some new abstractions have been added, but the underlying model is still the three-tier client-server-database architecture [Murugesan07].

Being just a large collection of static interlinked hypertext documents, the Web has traditionally had no high performance or real-time requirements. The typical workflow is that a user clicks a link, waits for a page to load, reads it, and finally starts the same workflow again by clicking a next link. With a web application, the workflow is different. The user interface should remain responsive while the client-side application constantly communicates with the server.

In a way, all the web applications are multi-user systems as multiple users can access and use them simultaneously. However, in this thesis a multi-user system refers to a system where users interact. For applications with slow asynchronous interactions, such as email, social media, blogging, and online shopping, the Web offers a well-suited platform. However, in multi-user web applications with real-time interactions the situation is more complex. In context of this thesis, the real-time does not indicate hard real-time constraints [Liu73] but fast interactions between the users as defined by [Ellis89]. In the real-time multi-user applications, the several users are using the system at the same time and are interacting with each other using a browser. The most typical examples are online chats, multi-player games, and collaborative editors. We speculate that the real-time multi-user features are currently getting more common. For enabling these features, the conventions for developing web applications need to be changed.

1.1. Motivation and Background

Douglas Englebart demonstrated the first collaborative real-time text editor in the year 1968 in “The Mother of All Demos” [Engalbart68]. Multi-player games working on the predecessors of Internet also existed, the most famous example probably being Maze War from the year 1973 [Colley04]. The earliest examples of collaborative native applications in the Internet date back to the 80s. Mail Transition Plan [Cerf80] from the

year 1980 describes an email system working over the Internet. [Richman87] underlines the importance of groupware by saying: “Like an electronic sinew that binds teams together, the new groupware aims to place the computers squarely in the middle of communications among managers, technicians, and anyone else who interacts in groups, revolutionizing the way they work.” Soon after the email, the first non-demo real-time multi-user online applications emerged. Some of the first examples were Internet Relay Chat (IRC) [Oikarinen13], an early form of instant messaging, and Multi User Dungeons (MUD) [William02], which are ancestors of massively multiplayer online role playing games (MMORPG) like World of Warcraft [WoW13]. After the first examples, lots of multi-user desktop applications and games, which are using Internet as a communication channel, have been created.

All the above examples have dedicated desktop client applications. However, [Taivalsaari11] claims that web applications offer significant benefits over native applications. For example, the delivery is simpler as they are instantly globally accessible for all browsing capable devices after deployment. The same applies to adding new features and bug fixes: the application running on a server is updated and the modifications became immediately available for all of the users [Jazayeri07]. Especially in the world of mobile devices, it is challenging to develop native applications as the number of different operating systems, versions, screen sizes, and additional capabilities has exploded. Writing code just for the Web instead of multiple separate device types reduces development costs. Differences between browsers exist, but they seem more likely to become standardized than operating systems. Furthermore, JavaScript libraries are hiding some of the differences. Thus, the old “write once, run anywhere” slogan [Wong02] is finally coming true.

Web applications excel in many ways desktop applications, but also pitfalls and obstacles exist. The technology stack of the Web is in many ways outdated and originally not intended for the complex use cases of today. For example, the HTTP request-response pattern is used for interacting with the server, and therefore the communication is always initiated by the client. When considering real-time communication between the users, native applications can take a direct peer-to-peer connection by using sockets and send messages to both of the directions. In a browser, this is not so straightforward.

The Web is strongly rooted in information sharing, and the current technical solutions rather isolate users into private sessions than make them aware of each other. Furthermore, many frameworks use databases for storing and sharing data and tend to require refreshes for showing new data. [Edwards97] discusses the strengths and weaknesses of asynchronous collaboration like email, group calendars, and knowledge databases. The database creates an additional gap that has to be overcome when developing real-time multi-user application.

Web applications are distributed applications by their nature and always divided between at least two components. The first is a client that consists of a web browser and a web page containing the user-interface and the client-side logic. The second is a server that is a computer with the stack of software running somewhere in the Internet. Traditionally, the majority of the logic has been on the server, and this we call a thin client approach. In contrast, if logic is on the client-side, then after exceeding a certain limit the approach is called a thick client approach. In addition, it is possible to implement native applications that act as clients and they are referred with a custom client term.

1.2. Research Problem and Aim

In this work, the main contribution is that we explain and analyze the current model of developing web applications. We are pinpointing problems that hinder developing multi-user real-time web applications and identify solutions to circumvent them. Furthermore, we are aiming to extend and describe the solutions at the abstract level. The included publications describe some more concrete examples.

Thus the main research question of the thesis can be formalized as follows:
How to easily develop web applications with real-time multi-user features?

To answer the main research question we have divided it to on several sub-questions:
Q1: *How to enable communication?*
Q2: *How to enable multicasting?*
Q3: *How should the application be partitioned between the client and the server?*
Q4: *How do development tools and frameworks support the development process?*

The answers to these questions are summarized at the conclusion.

This thesis focuses on a developer point-of-view, actual implementations, and technologies that enable real-time multi-user applications. The same theme could be studied from other perspectives, say, the business or user centric perspective, to name a few, but these fall beyond the scope of this thesis. We do not suggest any technological improvements on protocols or web browsers as they are slow and difficult to realize and standardize globally. As we are concentrating on the solutions and combinations of technologies already available in most web browsers, they can be exploited immediately in concrete application examples.

1.3. Introduction to Included Publications

The contributions of the research in the included Publications are divided as follows.

First, Publication [I] discusses how the development of MUPE (Multi-User Publishing Environment) applications could be simplified. The MUPE framework enables rapid

development of real-time multi-user mobile applications and an experienced developer can use it for easily building complex applications. However, because of the client server architecture and proprietary XML format, the learning curve is quite steep. To reduce the learning curve, we implemented a toolset to help new users and for helping experienced developers to complete MUPE applications even faster than before. The toolset was implemented as a plugin for Eclipse IDE [Eclipse2013], and it automates tasks like creating a new MUPE project and makes some tasks like editing the proprietary XML format easier. For testing the capabilities of the toolset, an example game, Dung Beetle, was implemented. The author has written most of the article and designed and implemented most of the plugin components presented.

In [II] some MUPE applications implemented by beginners and more experienced developers were introduced and analyzed. Our experiences backed up the assumption that by using the Eclipse plugin introduced in [I] it is possible even for beginners to implement complex MUPE application. Code quality metrics were used for analyzing the projects, and a conclusion was drawn that the quality of applications made by beginners was quite high. Reasons for this were application skeletons, editor, and snippets offered by the toolset. The author has written about half of the article and implemented Sandman and Dung Beetle used as examples.

[III] presents a complex real-time multi-user game called Sandman. Sandman is implemented using MUPE, and it is a bit like a tag game enhanced with mobile devices. The game uses Bluetooth to measure player proximity, and is played physically by running, chasing, and evading. Furthermore, the game contains a mechanism for grouping the users and calculating the winning side. The game was tested with several user groups and, based on these test games, some analysis on the game was done. The author has designed and implemented the application described (excluding the graphics). Field tests were also monitored and mostly organized by the author. In addition, he has written half of the text.

In [IV] we moved from custom clients and mobile devices to the browser and tried to implement the same multi-user real-time experience for the browser. For implementing the client-side features, we used the Lively Kernel [Taivalsaari08] that is a desktop-like environment running inside a browser. On top of the Lively Kernel, we built a real-time multi-user environment and example multi-user applications. The author has designed and implemented the client-side features excluding the communication parts. In addition he has written half of the article.

Publication [V] introduces a browser-based IDE called Arvue that uses CoRED as its code editor. CoRED is described more in detail in [VI]. Arvue is intended for creating and hosting Vaadin applications. In addition to CoRED, Arvue contains a graphical editor for editing user interfaces, connections to Git version control system [Git13], and an auto scaling framework for a scalable hosting of web applications. In addition, security concerns attached to hosting multiple untrusted applications are introduced and

partially solved. The author has implemented a part of CoRED that has been used as the code editor of Arvue, and has participated in writing the article.

For Publication [VI] we have developed a browser-based multi-user real-time code editor called CoRED. CoRED is implemented with Vaadin, and can be used as an editor component in web-based IDEs like Arvue described in [V]. CoRED contains tools for collaboratively editing Java code and Vaadin applications. Furthermore, it offers highlighting, indentations, semantic error checking, and code completion, which is a toolset often available in desktop based tools, but virtually unknown in browser-based systems. The author has implemented some of the server-side functionalities like resolving code completions and checking the errors. The author has also written half of the article.

In [VII] a browser-based multi-player role-playing game example implemented with Node.js [NodeJS13] is presented. The book chapter concentrates mostly on the dynamic content generation in a computer role playing game but also introduces how NowJS [NowJS13] can be used for enabling multi-player features in browser-based games. The author has written the parts concerning web applications and multi-user paradigm and participated in writing the other parts.

1.4. Scope of the Research and Methodology

In [Johansen88], the topology of groupware systems is presented. In the original presentation, the systems are divided into a 2-by-2 matrix (see Table 1), but later many modifications like a 3-by-3 matrix in [Grudin94] have been presented. Our work concentrates on **the same time and the different place** quarter of the Table 1. The application types in our quarter include video conferencing, instant messaging, chats, MUDs, virtual worlds, shared screens, and multi-user editors. Most of the current and traditional web-based collaboration sites are located in the different place and time quarter but we speculate that a trend towards the same time exists.

Table 1. 2-by-2 matrix by Johansen

		Time	
		Same	**Different**
Place	**Same**	Face to face interactions *(decision rooms, single display groupware, shared table, wall displays, roomware, …)*	Continuous task *(team rooms, large public displays, shift work, groupware, project management, …)*
	Different	Remote interactions *(video conferencing, instant messaging, chats/MUDs/virtual worlds, shared screens, multi-user editors, …)*	Communication + coordination, *(email, bulletin boards, blogs, asynchronous conferencing, group calendars, workflow, version control, wikis, …)*

In [Järvinen00] the taxonomy of research methods for studying information systems is presented. This thesis concentrates on man-made artifacts by evaluating the earlier approach, and after that, by building new ones. In this thesis we do evaluation on the Web of today, on real-time multi-user native applications, and on some applications and frameworks available in the Web. We use our findings to construct new applications and to outline the changes that should be made for making the Web and frameworks more appealing for real-time multi-user applications.

[Nunamaker90] proposes that system development and empirical research methodologies are complementary to each other. Furthermore, the author claims that system development, especially the development of a software system, is a research domain as well as a research methodology. In airplane design, starting from the Wright brothers and in many other fields the research progress has been: 1) Build a system, 2) develop theories and principles for observing behavior, 3) encode expertise in tools for easy access, and 4) use these tools to help the development of a new system. Most of the Publications made in context of this thesis fall in the first phase, which contains the design, development, analysis, observations, and evaluation of a software system. In the proposed taxonomy, this thesis falls in the second category and hopefully challenges the processes and concepts in a domain and expands the horizons of human knowledge.

1.5. Organization of the Thesis

The introductory part of this thesis is organized as follows. Chapter 2 elaborates the background regarding the Web and web applications. Chapter 3 shows the current situation for browser-based multi-user web applications. Chapter 4 discusses findings and lessons that we have come by during the work. Chapter 5 contains related research and some comparison of different approaches. Chapter 6 completes the introduction with concluding remarks, reviews the research question, and elaborates different possible branches of future work.

2. WEB APPLICATIONS

The World Wide Web (WWW) has evolved from a static collection of hypermedia pages to a platform for dynamic web applications. The promise of web applications is to be globally accessible in a device-independent fashion. Furthermore, they require no installs or updates; an up-to-date web browser is enough. The current trend is that web applications are replacing their native counterparts as the web applications are becoming more practical to use. [Taivalsaari08]

Earlier, the most web applications were implemented using plugin-based approaches such as Flash [Adobe13a], Java [Java13], Microsoft Silverlight [Microsoft13], Shockwave [Adobe13b], and Quicktime [Apple13]. The problem with plugins is that they are not universally available and require installing and updating. The current trend is that proportion of rich internet applications (RIA) implemented using standard web technologies like HTML, CSS, DOM, and ECMAScript is growing, whereas plugins based solutions are becoming increasingly rare. [Taivalsaari08]

2.1. Basics of the Web

The Web is based on relatively simple technologies described in W3C recommendation [Jacobs04]. Even though in the illustration the technology stack of the Web (Figure 1) seems clear and well-founded, practice is more complicated. [Mikkonen08] discusses how the current technologies make it possible to build web pages that behave much like desktop applications, but the development environments and tools have not yet adapted to this paradigm shift. In the desktop world, it is possible to implement complex applications with a single programming language. In the Web, different syntaxes are commonly mixed in a single file and several ways to implement the same functionality exist. The following summarizes the basic building blocks of web applications.

Figure 1. Technology Stack of the Web

eXtensible Hypertext Markup Language ((X)HTML) [Raggett99] defines the basic structure of a web page. The declarative nature of HTML makes it an easy language to write or generate. For defining a simple static web page, a single HTML file is enough, and CSS, ECMAScript, or binary data like images can be used for enchasing the page. For making easily reusable code, additional content is often introduced in separate files, but non-binary formats can also be easily embedded in an HTML file as well.

Cascading Style Sheets (CSS) [Bos11] is a language that allows authors to control style (e.g., fonts, spacing, and color) of HTML elements. Prior to CSS, HTML markup was used for presentational attributes. By using CSS, that information can be moved to another file, resulting in simpler HTML code. More recently introduced CSS3 offers interesting new features like visual effects, transformations, transitions, and animations [McFarland12].

ECMAScript [Ecma11] is a prototype-based object oriented scripting language used for implementing the dynamic client-side content in the Web. [Paulson07] declares that a trend from compiled to dynamic languages exists, and explains why the dynamic languages boost productivity. Several different ECMAScript dialects exist like JavaScript, Jscript, and ActionScript. We have used JavaScript and, therefore, later in this thesis we use term JavaScript instead of ECMAScript. Use of JavaScript as a programming language has been analyzed in [Mikkonen07]. Furthermore, [Crockford08] presents good and bad characteristics of JavaScript. As an alternative for JavaScript some plugin-based approaches like Flash, Java, or Microsoft Silverlight can be used for implementing the client-side logic.

Extensible Markup Language (XML) [Bray06] and **JavaScript Object Notation (JSON)** [Crockford06] are text-based data formats that are both human and machine readable. In the Web, the formats can be used for language independent messaging and for exchanging representations or data. Libraries for manipulating the formats exist for a number of programming languages.

Document Object Model (DOM) [Wood98] is a platform neutral interface that allows scripts to dynamically access and alter the content, structure, and style of a web page. While rendering an HTML document, the browser assembles the elements included in the document to the DOM that acts as a shared data structure between a browser and a JavaScript application.

Uniform Resource Identifier (URI) [Berners-Lee05] provides simple means for identifying and addressing different types of a resource. In the Web, the resource can be for example (X)HTML, CSS, or JavaScript formatted document but some media types like images and video are present as binary formats [Freed96]. For instance, *http://www.example.com/* is a valid URI. The beginning of a URI defines the protocol to use, and the rest identifies the hostname for the resource.

HTTP Hypertext Transfer Protocol [Fielding99] is a protocol that is used in the Web for transferring resources. HTTP is a generic, stateless request-response protocol that follows the client-server computing model. In modern web applications, the demands have changed from what was adequate in the static Web, and sometimes consecutive requests are combined to sessions or different protocols are used for enabling bidirectional communication.

2.2. Architecture of Web Applications

The architecture of a web application often follows the three-tier architecture (Figure 2). The tiers are named as presentation logic, business logic, and data logic [Peacock00]. To roughly connect the theory and practice, the presentation logic runs on the web browser that acts as a client (Tier 1), the business logic runs on the web server (Tier 2). Furthermore, a database server is used for enabling persistence (Tier 3). The communication between the tiers is pull-based: upper layers always initiate the communication by asking for data from the layer directly below. For pulling, the browser uses HTTP for requesting new updates from the server, and the server uses queries (arrow between the middle and right box Fig 2) for getting the related data from the database. In this thesis we often use terms: client-side (a web browser), server-side (a web server), and database because they more concretely reflect the structure of the implementation.

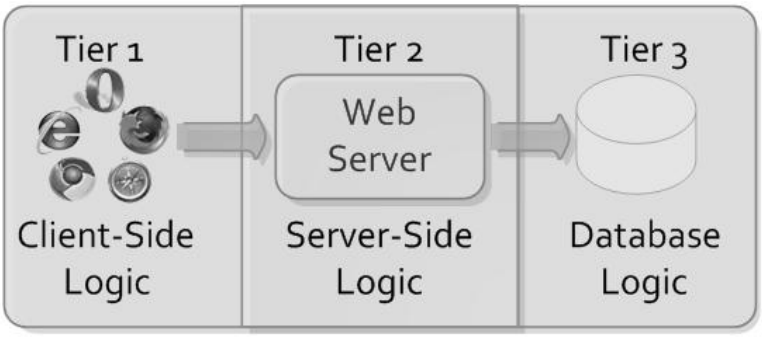

Figure 2. Three-tier architecture of web applications

In static web pages, HTML and CSS describe the user interface (UI), and if application logic is needed, it runs on the server-side. Each interaction causes a request that is responded with a new HTML document. The approach works well for hypertext and some form-based applications, but is hardly acceptable for most web applications, as the full page reloading is time consuming and not very user friendly. The current trend is that **client-side** (Figure 2 left) contains a JavaScript client for communicating with the server. JavaScript virtual machines have become increasingly efficient and rich in features, and this makes it possible to transfer more of the business logic to the client-side. The evaluations in [Anttonen11b] show that technologies have matured and are now ready for use. For implementing user interfaces and client-side functionalities, the JavaScript libraries [Dojo13] [Bibeault08] [Prototype13] are often utilized.

Sometimes the standard technology stack is not flexible enough or does not offer the APIs needed. Custom clients and browser extensions can be implemented to avoid these restrictions. Currently, many mobile devices have their own custom applications for accessing Facebook, email, or weather services, for instance. Custom clients, commonly referred as apps may, for example, enable an effortless picture upload or GPS for tracking. These clients can be adapted to the special features of the device, but they lack the versatility and universality of the browser approach.

Server-side (middle in Figure 2) of a web application consists of a web server and a web application. For hosting a web application, a developer does not typically need to alter the web server, but just configure it to find and serve the application and other content.

A web application framework (WAF) is often used for implementing the server-side logic of a web application. [Shan06] defines that WAF is a reusable, skeletal, semi-complete, and modular platform that can be specialized to produce custom web applications, which commonly serve the web browsers via the HTTP protocol. The frameworks simplify development considerably as they hide parts of the technology stack mentioned earlier. Contrary to the restricted set of programming languages available for the implementing client-side features, there are numerous frameworks and languages including Python [Rossum94] and Java [Java13] can be used for developing the server-side of a web application. However, by selecting a web application framework like Node.js [NodeJS13] or Vaadin [Grönroos13], the language is often predefined by the framework.

The third tier (Figure 2 right) of the architecture is the **Database** tier. For a simple web application this tier is not necessarily needed, but often it is inevitable. Lots of different types of databases for different needs are available, but basically they are all used for persistently storing and logically organizing the data. Alongside easy UI creation, the trend in web application frameworks is to aim at as easy database connectivity as possible.

In a web application, the HTTP protocol is used for implementing **communication** between the client and the server (arrow between the left and middle boxes in Fig 2). The relationship between the server and clients is asymmetric. The client initiates communication by sending a request consisting of the URI, the method, and some other headers to the server [Fielding99]. The server responds with a status line, header, and payload data. In the original approach, after each user interface event, the whole document was always downloaded and rendered. In the context of web applications, this kind of behavior would be time and bandwidth consuming and would reduce the responsiveness. The performance can be dramatically improved by updating only parts of the user interface using the Ajax framework [Crane05, O'Reilly07]. Originally, Ajax stood for Asynchronous JavaScript and XML, but today the term is widely used for

other programming languages and notations than JavaScript and XML. In JavaScript, libraries are commonly used for enabling the Ajax communication.

2.3. Partitioning the Application

Distributed applications consist of computer nodes that are communicating over the network for achieving a common goal [Andrews00]. As a web application consist of at least the server and clients and use HTTP for communicating over the network, they can be seen as a form of distributed computing. As the application is divided between the computing nodes, it is necessary to select which component takes responsibility of what functionality. [Zhao02] criticizes ad hoc solutions that have created infrastructure that makes implementing, upgrading, and maintaining a web application very complicated. Furthermore, they make an observation that request-reply nature of HTTP creates a difficult segmentation between the execution of application and user interface. In their follow-up paper [Zhao03], the authors concentrate more on the software architecture of web applications. They divide the application in accordance with the three-tier architecture like in Figure 2, and make an observation that there is a problem in mapping the traditional functionalities (presentation, application, and database logics) on the tiers. The proposed approach is to split presentation logic across the HTTP channel so that the client contains part of it and the server contains the rest.

Currently, the clients of web applications seem to be getting thicker. That is mostly due to the adoption of scripting languages in general, Ajax, and Flash [O'Reilly07]. [Kuuskeri09] proposes a partitioning that aims at moving most, if not all, of the application logic to the client-side. The need for the server still exists, at least for serving the application, managing the database, and for enabling communication between the clients, but the JavaScript application itself can run inside the browser. This approach enhances the responsiveness and robustness of the user interface, and makes the development much more straightforward because the separation of concerns is clear: only one language, in this case JavaScript, is needed for implementing the application.

In the context of this thesis, thick and thin clients represent two extremes. In the thick client approach, the application logic and the state are stored at the client-side. In the thin client approach, the application, including its state, runs on the server-side and only the user interface is shown to the client. The thick client approach utilizes the processing power of a modern PC, and in some cases, the client can run independently after its initial download. For devices with limited processing capabilities, the thin client approach may be a better option. In addition, it is important to notice that users are able to view and modify the source code of a JavaScript based thick client and, therefore, some precautions should be made. Furthermore, in approaches that base on HTML and CSS, graphical designers have been able to partially generate the pages, but with the thick client approach it is more difficult for non-programmers to participate in a project.

RESTful web services are currently becoming more popular in the field of web applications [Richardson07]. In a RESTful web application a consumer is responsible for maintaining application state and logic. The RESTful service stores data and offers URIs and representations for the resources. The requests from the consumer contain all the necessary context information for responding and therefore the server does not need to maintain the state of the client. As the consumer of a RESTful interface needs to be able to store its own state, a thick client approach can well be utilized. When combining the thin client approach and a RESTful interface, the connections to the interface can be done in the server-side.

In general, no partitioning is universally the best; the different approaches fit to different purposes. In most of the frameworks, the application logic is somehow partitioned between the client and server. The important thing is to aim at a good separation of concerns based on the current needs and design which parts of the application are where and why. To make the concept clear, let us consider a simple drawing application. In the case of a thick client, the whole drawing application is implemented with JavaScript and an initial version of image is downloaded from the server at the beginning, the editing is performed locally, and finally the image is uploaded back to the server. In a thin client approach, only the user interface and visible parts of the image are downloaded at the beginning. During the editing, the requests are sent to the server that updates the drawing and then returns the modified image as a response. Both of the approaches have their pros and cons. With the thick client, the user interface of the application is fast and responsive, and after the startup the amount of communication is minimal. As a downside, more data has to be downloaded at the beginning and system may start up slowly. Furthermore, the image processing may be resource consuming. In the thin client system, minimal amount of data is downloaded at the start, but more bandwidth is needed during the editing. The thick client system could possibly be used even in offline mode. Naturally, this is not possible in the thin client approach.

2.4. Web Application Frameworks

A web application framework is an application framework that can be used while developing a web application. The aim, like in all application frameworks, is to facilitate the development process. In case of web application frameworks it is possible to offer libraries, template engines, session management, and other solutions that can be reused in web applications of all kind. Furthermore, the frameworks are hopefully tested in different projects and developed by professional and, therefore, the applications utilizing a framework are more secure and less prone to errors.

In the following subsections, some web application frameworks are introduced. At first we present a discontinued custom client example MUPE (Multi-User Publishing Environment) [Suomela04]. MUPE was not based on a standard web browser, but

followed the client-server architecture and offers good support for the real-time multi-user applications at framework level. After MUPE, we present the Lively Kernel [Ingalls08, Taivalsaari08] and Vaadin [Grönroos13]. Frameworks are selected because they offer the good examples of a thick and a thin client based approach. However, they do not represent the mainstream of web application frameworks and, therefore, we also introduce JavaScript based Node.js [NodeJS13]. In Node.js and MUPE, the thickness of a client is related to selections made by a developer, whereas with the Lively Kernel and Vaadin the thickness is predefined.

2.4.1. Native Web Client – MUPE

MUPE (Multi-User Publishing Environment) [I, II, III, Koivisto03, Koskinen06, Suomela04, and Suomela05] was a client-server environment for implementing multi-user applications and games for Java ME-based [Oracle13a] mobile devices. MUPE server was built on top of Java and client-side on top of Java ME. MUPE server was a package that contains MUPE Core, Context Manager, and MUPE Service that can be extended to a new MUPE application. The high-level architecture of the MUPE framework is presented in Figure 3. In the first versions of MUPE, HTTP requests and responses were used for realizing the networking between the client and Core, but for enabling the bidirectional communication, the implementation was changed to use TCP sockets. Furthermore, Context Manager was used to provide external information, such as weather forecast or XML feed, to the service [MUPE08]. Instead of HTML, the framework used a specific XML-based language for defining the user interfaces and client-side functionalities. However, during the latter phases of the framework development some experiments with client-side JavaScript were made.

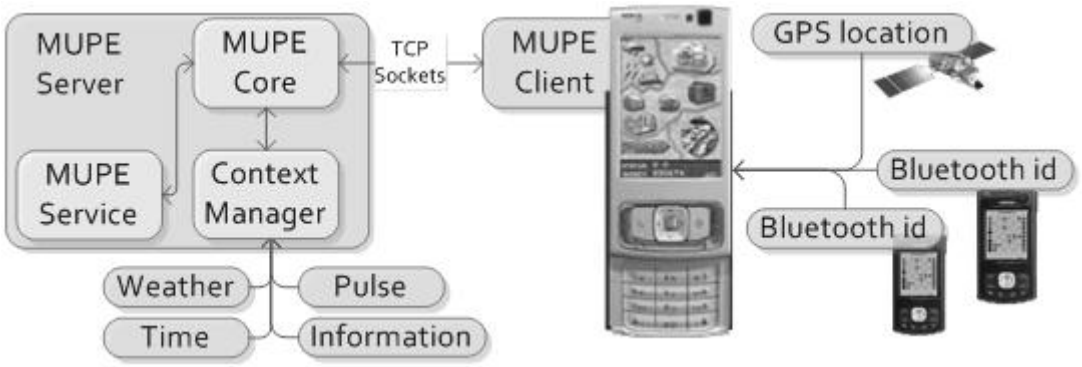

Figure 3. Contexts and the MUPE framework

Initially, MUPE client, which, in essence, is a special-purpose browser for browsing MUPE applications, has to be installed on the mobile device, but after installation it can be used to browse and access all MUPE applications. In MUPE, the developer is somewhat free to choose how the application logic is partitioned between the client and server. However, the most straightforward approach is to implement most of the logic at the server-side using Java and then UIs and simple client-side functionalities using XML scripting. To support persistence, any methods commonly used in Java can be used.

As a custom client, MUPE offers features that are not available for a standard browser. For example, it has permissive access to the device APIs like camera or GPS module. Therefore, the features of the device can be used more freely as a component of the application. Furthermore, MUPE offers bidirectional TCP socket communication between a client and the server. For applications like image processing MUPE could easily use the camera of the device, and GPS location could be added to the picture.

2.4.2. Thick Client Framework – The Lively Kernel

The Lively Kernel [IV, Ingalls08, Taivalsaari08] is a web application framework written entirely using JavaScript. For application developers, the Lively Kernel provides a rich set of user interface widgets, an event model, and a number of other useful features such as modules, classes, and networking. New applications can be implemented by extending JavaScript prototypes and by following certain Lively Kernel specific conventions. In Figure 4, a block structure of the components of the Lively Kernel framework is presented. The Lively Kernel box contains all the libraries, tools, and UI components that the Lively Kernel offers. The Lively Kernel is built on top of a JavaScript library called Prototype [Prototype13]. The components within solid black line are provided by the browser [Sun08]. Developers implement new Lively Kernel applications using JavaScript and the applications are completely executed on the client-side. Several applications can be run, and they all remain active while another application or networking is used.

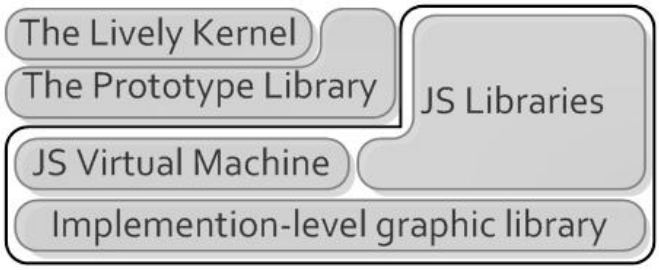

Figure 4. High-level block structure of the Lively Kernel.

A Lively Kernel application does not necessarily need a server except for bootstrapping the system. After the initial startup, the system can be run independently without networking, if no active applications need further communication. XMLHttpRequests [Kester07] can be used for communicating with the server. In the Lively Kernel example applications for viewing the weather and stocks contexts are implemented.

The same origin policy [Ruderman01] prevents direct downloads of HTML from different domains, but in the Lively Kernel the problem was circumvented by utilizing the security bug available in certain version of Safari web browser. The more general approach to circumventing the same origin policy is the use of a server-side proxy or specific technologies like CORS [Kester10] or JSONP [Ippolito05]. For implementing the server-side the developer is free to select the language [IV]. Often a simple JSON

database like Persevere [Persevere13] that can be manipulated using HTTP/REST [Richardson07] interface may be enough.

2.4.3. Thin Client Framework – Vaadin

Vaadin [V, VI, Grönroos13] web applications are implemented almost like any Java Standard Edition desktop applications; the most obvious difference is that instead of Java GUI widget toolkits like Swing [Eckstein98] or AWT [Zukowski97], the Vaadin applications use a custom set of GUI components. The developer can implement server-side applications using the full power and flexibility of Java, and the business logic of the Vaadin application runs on the server-side. Vaadin is capable of automatically rendering the UI components over the network on any browser, and as a user interacts with the components, the Vaadin framework automatically uses AJAX for non-blocking communication with the business logic running on the server. Furthermore, the session management is offered automatically by the system.

Vaadin consists of the server-side framework and a client-side engine. The client-side engine renders the user interface components and communicates over HTTP with the server. In Figure 5, the components inside the big light rectangle belong to Vaadin framework. In most cases, the only parts a developer needs to touch are the business logic of the Java application (second from the right box in Figure 5) and the connections to selected services (boxes on the right-hand side of the Figure 5). If the need for custom made client-side components arises, they can be implemented by using Java, JavaScript, and Google Web Toolkit (GWT) [Google13a, Perry07]. For storing permanent data in a Vaadin application, Vaadin offers the Persistence module, but any other solution including databases can be also used like in any other Java application.

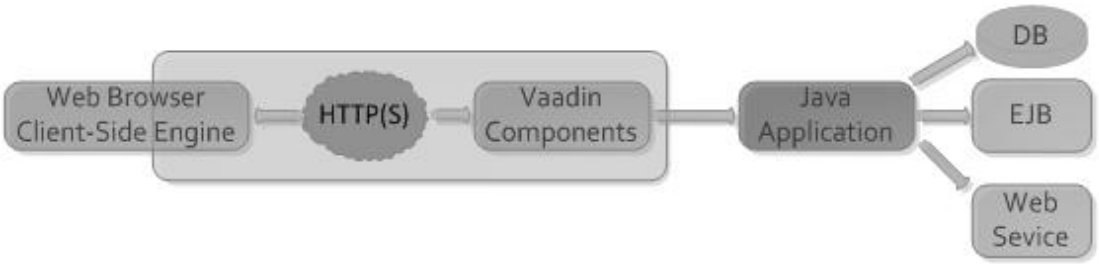

Figure 5. The basic architecture of vaadin framework.

2.4.4. Mainstream Framework – Node.js

Node.js [VII, NodeJS13] is a platform built on the V8 JavaScript engine [Google13b]. It is intended for developing JavaScript-based network applications. By using libraries like Express [Express13], it can be extended to more complete web application framework. Node.js allows using JavaScript at both client and server-side and the developer is free to choose how the applications are partitioned between the client and the server. Furthermore, it is up to the developer to select how the client-side application communicates with the server. Data interchange formats like JSON or XML can easily

be used. Lots of additional libraries like Mongojs [MongoJS13] and Socket.IO [SocketIO13] are available for making development easier. Mongo.js allows user to use the Mongo database [Mongo13] and Socket.IO can be used for realizing WebSockets [Hickson11] communication between the client and the server.

2.5. Summary

In this chapter, we explained the technology stack behind web applications. We described the three-tier architecture that is by our best knowledge the most usual architecture for implementing web applications. The three tiers are client-side, server-side, and the database, which all have their own responsibilities. The database is used for storing the data, the client-side presents data to the user, and the rest of application logic is somehow divided between the client and the server depending on the framework and architectural selections. The issues regarding partitioning the application between the client and server are discussed and some pros and cons were listed. Furthermore, we also listed some notes about communication between the client and server.

The main findings of this Chapter are summarized in Table 2. In that we present the frameworks and discuss on how the separation of concerns is made. The separation is important as it sets boundaries on how the applications are implemented. In the next column, we describe what is special in the framework and in the final column we list where the framework has been used and explained with more details.

Table 2. Findings from example frameworks summarized

Frame-work:	**Separation of concerns:**	**Notes:**	**Utilized in Publication**
MUPE	Developer makes the separation	XML is used for scripting and describing user interfaces on the client-side. Java is used on the server-side	[I] to [III]
The Lively Kernel	All the application logic on the client-side	The thick client-side engine is implemented using JavaScript. Server-side is needed only for bootstrapping, communication and persistency	[IV]
Vaadin	All the application logic on the server-side	GWT-based client-side engine and UI components. The thick server-side is implemented using Java. Listener methods of the UI components located on the server-side.	[V] and [VI]
Node.js	Developer makes separation	JavaScript is used both on client- and server-side.	[VII]

3. MULTI-USER APPLICATIONS IN THE WEB

While Web 2.0 can be regarded as a marketing term, the applications still differ from the original Web by often allowing users to interact with other users [Taivalsaari08]. As typical examples, one can mention user generated content in Wikipedia, social media, and blogs. In addition, commenting and rating of news, blogs, images, articles, and purchases occurs in many different forms. The common nominator between the examples is that the users mostly work in their own private sandboxes but the outcomes are shared with the community.

3.1. Sample Multi-User Systems

The most of Web 2.0 systems offer multi-user features and [Kaplan10] makes division to different application types based on self-presentation and social presence. In the following, some common types of multi-user web applications are explained, and to make things more concrete some widely used web applications have been presented as examples.

Wiki pages are currently one of the most common forms of collaborative projects. Wiki is a web site that allows users to create and edit its content. In the Wiki approach, collaboration is slow and asynchronous, but results generated via collaboration can be impressive. Wikis have been built for many different purposes: Wikipedia (*http://wikipedia.org/*) is an encyclopedia, Wiktionary (*http://www.wiktionary.org/*) is a dictionary, Wikitravel (*http://wikitravel.org/*) is for creating a worldwide travel guide, OpenStreetMap (*http://www.openstreetmap.org/*) is used for mapping the world, and so on. Furthermore, lots of Wikis has been built on specific themes like popular culture and hobbies. Memory Alpha (*http://memory-alpha.org/*) is Wiki for everything related to Star Trek and Wookieepedia (*http://starwars.wikia.com/)* concentrates on Star Wars alone. Finally, many projects and organizations use a Wiki as their internal communication channel.

Social Networking sites refer to the web pages like like Facebook (*http://www.facebook.com/*), Google+ (*http://plus.google.com/*), and Twitter (*http://twitter.com*) that enable users to connect by creating personal information profiles and do casual everyday socialization and interactions. Sites offer features such as posting messages, liking, commenting and recommending. Many other web sites like blogs and news have embedded the possibility to like or comment on their content using social media. In addition to casual social media, there are sites like LinkedIn (*http://www.linkedin.com/*) and Klout (*http://klout.com/*) for very specific purposes.

Many other sites, like YouTube (*http://www.youtube.com/*), last.fm (*http://www.last.fm/*) and geocaching.com (*http://www.geocaching.com/*), have included features associated with social networking.

Internet is full of single player **games**. Probably the simplest way to update a single player game to a multiplayer game is to add a high score table and, thus, make it possible for users to compete against the scores of other players. When user identities at the shared high score table are connected with the Facebook accounts things are already becoming interesting, even if the level of interactions is low. If we really want to create a multi-player game, then the players have to be able to interact inside a game. As an example, we can mention FarmVille (*http://www.farmville.com/*) that allows the player perform various farm management related tasks like planting, growing, and harvesting crops. Players can also invite their friends to their neighbors in the game and help each other in various farm related tasks. In Travian (*http://www.travian.com/*), the player is a leader of a small medieval village and develops ones realm while fighting with other players. In Urban Dead (*http://www.urbandead.com/*), the player plays a survivor or a zombie in the world after a zombie apocalypse.

In **web stores** like Amazon (*http://www.amazon.com/*) the commenting mechanism can be used for reviewing the products and for giving recommendations about products that could appeal for the taste of the regular buyer. On auction sites like eBay (*http://www.ebay.com/*), and Finnish Huuto.net (*http://www.huuto.net/*), recommending can be used for evaluating the reliability of the users.

3.2. Real-Time Sample Systems

Examples listed in Section 3.1 are multi-user but not real-time multi-user systems. In the following, the real-time systems are elaborated and the list of application types is extended with real-time sample systems.

In this thesis, real-time refers to a behavior that the actions made by one user are mediated to related users as soon as possible. [Ellis89] defines that in the real-time system the notification time must be comparable to response time. Roughly, this means that mediating message from user to another should not be much slower than sending a message to the server [Shen02]. [Fettweis12] analyses in more detail how human physiology defines the strict time constraints of what can be considered as real-time.

In comparison to conventional web applications, real-time multi-user web applications are more challenging to implement. As the speed of communication is ultimately limited by the speed of light, the modifications to the state are not immediately propagated from one user to all the other users. Furthermore, the technology stack of the Web may not allow the server to immediately forward the message. Lag in messaging causes conflicts as the perception of the data is slightly unsynchronized between the parties [Lamport78]. More lag leads to more conflicts. In a collaborative editor, for

example, a conflict exists if one deletes a sentence that another is editing. While designing a multi-user application the selection needs to be made if we want to use locks [Gray98] and wait for synchronizing the views or if the users have their own data running little out of sync. In the first alternative, the user experience may not be as smooth and interactions are more by turns based than continuous and in the latter case there is a risk that software does not know how to smoothly merge unsynchronized data.

Collaborative online **editors** allow people to edit the same document together using different computers. These are an interesting special case of the multi-user web applications. In many of the browser-based editors, other users can be invited for real-time collaboration sessions. The most obvious example is of course Google Drive [Google10] but other examples like Cloud9IDE (*http://c9.io/*), Codenvy (*http://codenvy.com/*), Codev (*http://codev.it/*), Firepad (*http://www.firepad.io/*), MobWrite (I) and SynchroEdit (*http://www.synchroedit.com/*) also exist. Furthermore, in [V] and [VI] we have introduced CoRED that is a browser-based real-time collaborative editor for source code. Publication [VI] elaborates methods for avoiding conflicts while editing text. In contrast to synchronous editing, version control systems such as CVS, SVN, and Git can be seen as the examples of asynchronous text editing.

Many of the current browser-based games seem to be asynchronous by nature, but in games there is huge potential for impressive demonstrations after the implementation of real-time multi-user applications gets easier and technologies such as WebGL [Marrin11] mature and impressive 2D and 3D become standard features [Anttonen11a]. In [VII] we describe a collaborative browser-based game and briefly introduce some others. [Nokia03] addresses differences between multi- and single-player games. When designing a framework for gaming, those differences should be accounted. Furthermore, proposals concerning network drops, latencies, and user communities can be broadened on all multi-user applications. Ultimately the same problems have to be tackled in all kinds of multi-user and concurrent applications [Ellis89] and finally, if chased long enough, we end up thinking about time, clocks and the ordering of events in distributed systems [Lamport78].

Finally, communicative web applications like chats and messengers could benefit from real-time features. For example, Apache Wave [Apache13], previously known as Google Wave, was an attempt to create a collaborative environment which would eventually displace email and instant messaging [Fried10]. The revolutionary feature of the system was that it worked differently if the users were simultaneously online. In the online mode, the users were able to see edits and messages in real-time. If the receiver was not available, then the system worked asynchronously by email and showed messages as the receiver came online. Wave was not adopted by masses, but the same basic idea, excluding visible editing, is currently available, for example, in Facebook messaging.

Situation awareness between the users is considered one of the key aspects of a distributed real-time multi-user application. To be able to work together users need to be aware of each other. At the simplest case, the awareness can be enabled by exposing a list of online users, but additional information may be useful [You04]. Often friend lists are used for allowing users to select the people they want to interact with [Ellison07]. Familiar names and maybe even pictures are needed to identify other users.

There are no reasons to assume that users would dislike real-time multi-user applications running inside the browser. In [Mogan10] three brainstorming real-time applications were offered to users; first of them was a native application, second was a plugin-based and third was a web application. As a result, based on the opinions of the users, the web application version seemed to be the most appealing. In addition, the collaborative web applications are not just appealing for the users, but based on [Chui12] can also provide huge annual savings for enterprises.

3.3. Implementation Examples

For demonstrating the differences between the frameworks and for explaining their effect on the implementations of real-time multi-user web applications we have selected four example frameworks. We use the Lively Kernel as an example of the thick client, Vaadin represents the thin client systems, and MUPE and Node.js are compromises between the two. In the following subsections, we use this fast-paced game called Dung Beetle (Figure 6) as an example and concentrate on how the separation of concerns differs depending on the thickness of the client. Furthermore, we explain the communication needs and describe how the data is synchronized between the players.

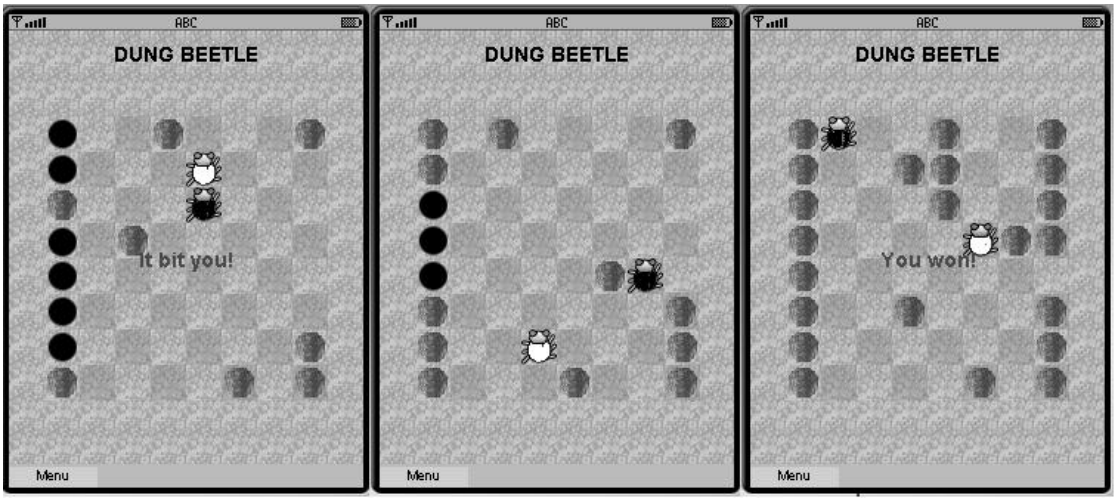

Figure 6. Three game situations from MUPE implementation of Dung Beetle

Dung Beetle, our example application, is a real-time two-player game where players are controlling their dung beetles that collect pieces of dung. Users act simultaneously and views are kept as synchronized as possible. At the beginning of the game, new piles of the dung start dropping on the gaming board randomly. Beetles collect them by pushing

dung towards their lairs on the sides of the gaming board. The winner is the beetle that first collects a pile of dung on each hole of its lair. Furthermore, it is also possible to bite an opponent. Biting paralyzes the opponent for a short moment. In Figure 6, the player controlling the black beetle is filling black holes with dung. In the leftmost picture, the white beetle has bitten the black and the black one has paralyzed for a moment. In the middle picture the game is going on normally. The black beetle has been able to collect five dung piles, and the white has only collected four. Furthermore, the black beetle is just pushing its sixth pile of dung towards the left. In the rightmost picture, the black beetle has managed to fill all the holes and has won.

3.3.1. Native Web Client – MUPE

MUPE was a framework for real-time multi-user applications and therefore many features like bidirectional communication, support for multicast, and policies for synchronizing the states were natively available. Compared with the other frameworks used in this thesis, in MUPE the implementation of the example game was the easiest. In Chapter 4, we discuss our lessons from MUPE and explain how to transfer some of its features to web application frameworks.

The MUPE version of Dung Beetle is not strictly synchronized between the clients, but each of the clients stores its separate view to the state. Figure 7 presents some typical game actions. At first, the server randomizes the locations of new dung piles and sends the notifications of new dung pile to the players. Then the first Player1 moves and new location is rendered. Then the movement notification is sent to the server that relays it to the Player2 which then updates the location of Player1. The client-side of application is implemented using XML scripting. Clients use functions like `clientMoveBeetle(int fromX, int fromY, int toX, int toY)` for communicating. In addition, similar methods for sending notifications about the position of dung piles and special action like biting and winning are used. Clients consume actions immediately after sending or receiving them.

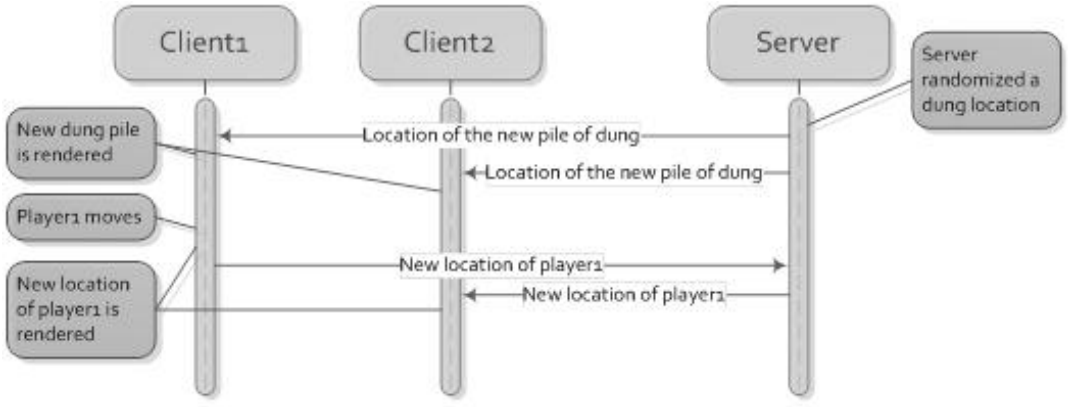

Figure 7. Sequence diagram of Dung Beetle for MUPE and Node.js

Depending on the game and the latencies, the selected approach could cause problems as the events are possibly executed in different time and order between the players.

Dung Beetle was designed to be tolerant to these problems, but some minor issues remain. For example, sometimes when the beetle bites another, from the perspective of the bitten beetle the biter has not visited the neighboring tile. This is because the lag causes movements to be executed in different order for the beetles. In addition it is possible to accidentally copy a piece of dung if it is pushed simultaneously in horizontal and vertical directions. If the game end is almost a tie, then both of the players may think they have won. However, if the players stop playing or play really slow, then states between the players would eventually synchronize.

3.3.2. Thick Client Framework – The Lively Kernel

Publication [IV] presents a set of real-time multi-user features implemented on the Lively Kernel. The set contain simple login capabilities, chat, friend list and Dung Beetle game. A screenshot from the multi-user Lively Kernel is presented in Figure 8. The left most item is a friend list that shows all the users who are online. By clicking any user, a chat panel opens and user private chat can be started. Our example game Dung Beetle is at right. The implementation of Dung Beetle was quite straightforward to do with The Lively Kernel, except the framework did not offer any support for making communication between the players nor the server-side and therefore some extra work was needed to implement the communication between the players.

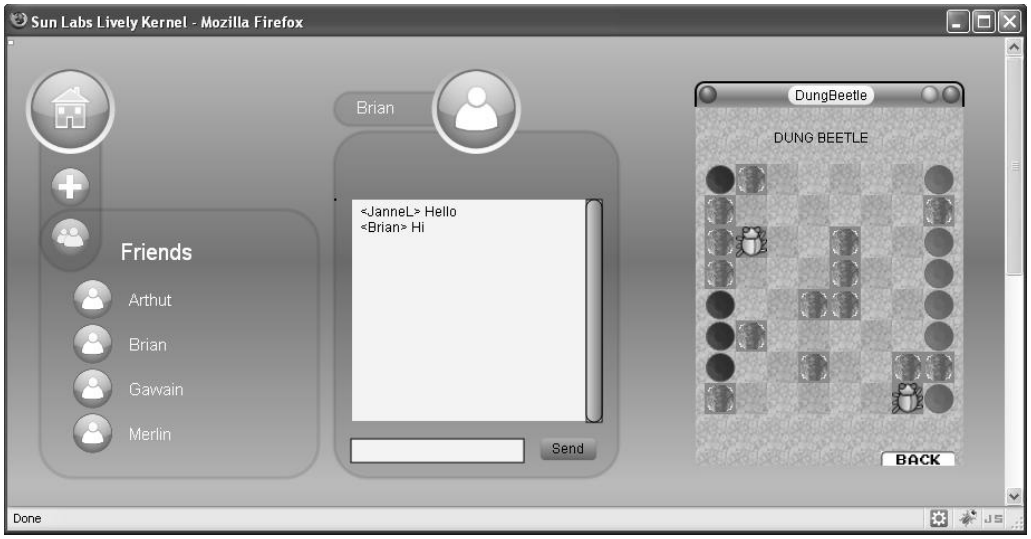

Figure 8. The Lively Kernel implementation of collaborative features

As an example of the extremely thick client, the Lively Kernel does not necessarily need server-side except for the initial download. Therefore, a small server layer is implemented for enabling the login capabilities and communication between the users. The communication is done using DojoX [DojoX13], an extension of Dojo toolkit that offers the server push feature. The simplified class diagram is available in Figure 9. All the applications, `LivelyChat`, `DungBeetle`, and `FriendListPanel` are positioned to `MainWindow`. Furthermore, all multi-user applications are connected to `Dojox` that is connected to the `Server`. Method `Subscribe()` is used for connecting a new initialized application with `Dojox`. Methods `PublishMessage()`

and `ReceiveMessage()` enable the communication between the users. Finally, `Leave()` disconnects the application.

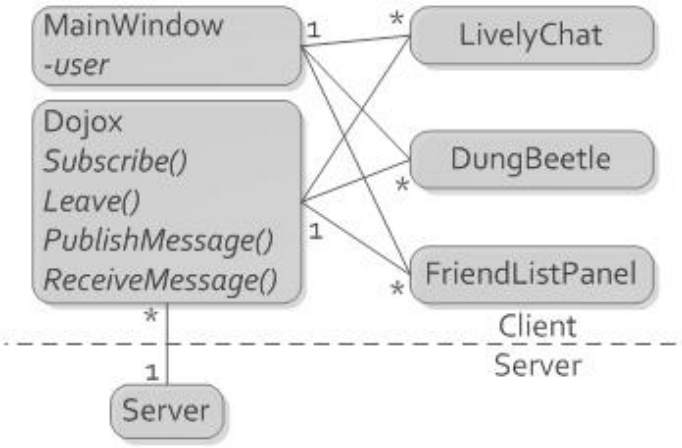

Figure 9. Simplified class diagram of the Lively Kernel implementation

In a Figure 10, sequence diagram is used for presenting some typical game actions. The diagram starts with and an event that randomizes a new dung location. This event is fired by a timer. Both of the clients are able to randomize new locations and this is done in turns. Immediately after the randomization, the dung pile is rendered on the screen of Client1. The server forwards the location to the Client2 which, after receiving the message, renders it on the screen. As a next event, a user of Client1 decides to press an arrow key and the beetle moves. The movement is rendered immediately on the screen of Client1 and then, after going over network, on the screen of the Client2. From the sequence diagram it can be inferred that the events are immediately consumed by the client that initiates them and on the other client. In the diagram, the actions are only fired by Client1, but in real game situations, both clients would constantly fire new events without waiting for the network or any responses.

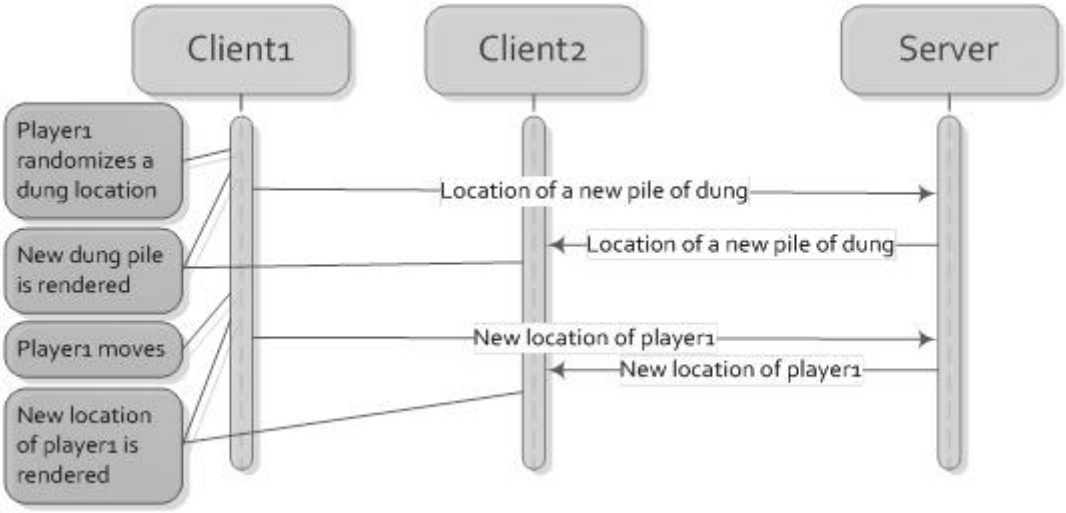

Figure 10. Sequence diagram of Dung Beetle for the Lively Kernel

In the Lively Kernel implementation of the Dung Beetle, the thick client approach is followed and the tasks are run on the client-side. The MUPE version does the

randomization of new dung pile locations on the server. With the Lively Kernel implementation, the task is more complicated. Technically, the locations could, of course, be randomized in the thin server layer, but this would be against the selected ideology. Another approach could be to promote one of the clients to the master client who would always randomize the location. This approach would benefit the master client as it would see a new dung ball immediately and other player just after network delays. Our solution was to randomize new locations for dung balls on both of the clients based on a randomized timer. Other features requiring communication follow the same patterns as in the MUPE version. Game runs unsynchronized and independently in the clients and the thin server layer is used for relaying messages between the instances. As the Lively Kernel Implementation follows the MUPE implementation, its problems afflict the Lively Kernel version, too.

3.3.3. Thin Client Framework – Vaadin

A Vaadin implementation of the collaborative features experimented in the Lively Kernel has also been implemented (Figure 11). The page is simply divided into four panels. On the top left panel, the user can login to the system and in the top right there is a list of online users. The bottom left panel contains the chat screen and in the bottom right there is Dung Beetle, our example game. As with the Lively Kernel, the implementation of the user interface of the game was quite an easy task. At the time when the game of the example was implemented, no guidelines enabling the communication between the players existed, but later a tutorial for broadcasting messages to other users has been added [Vaadin13].

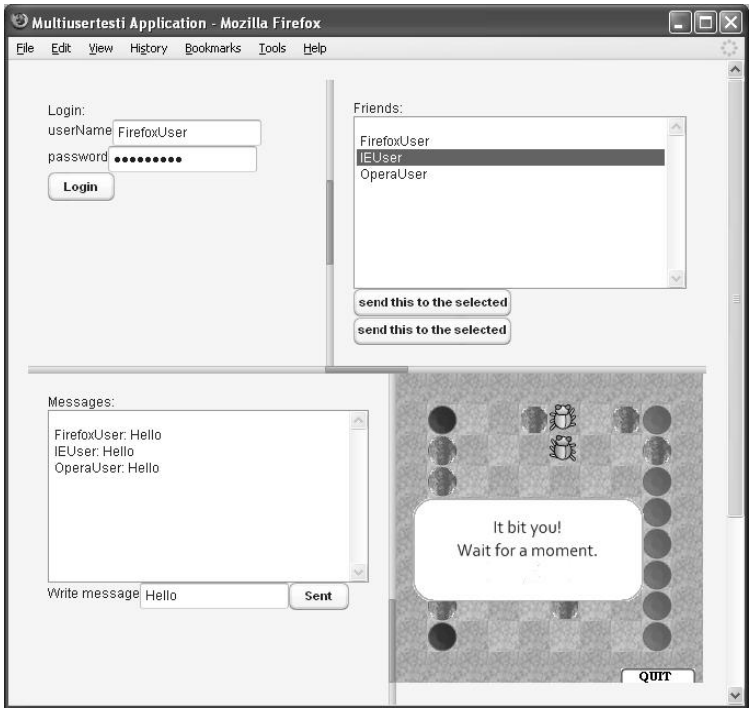

Figure 11. Vaadin implementation of collaborative features

The class diagram in Figure 12 presents a simplified structure of the system. In the Vaadin version of Dung Beetle, the game runs on the server and only the client-side engine common to all Vaadin applications is sent to the browser. For the browser, the game and the other panels appear as collections of GUI components like text fields, images, and buttons. User actions, like clicks and keyboard events, are sent to the server for processing and the client-side engine processes the responses and modifies the view accordingly. The server-side contains the game logic and knows how the components are related to the game.

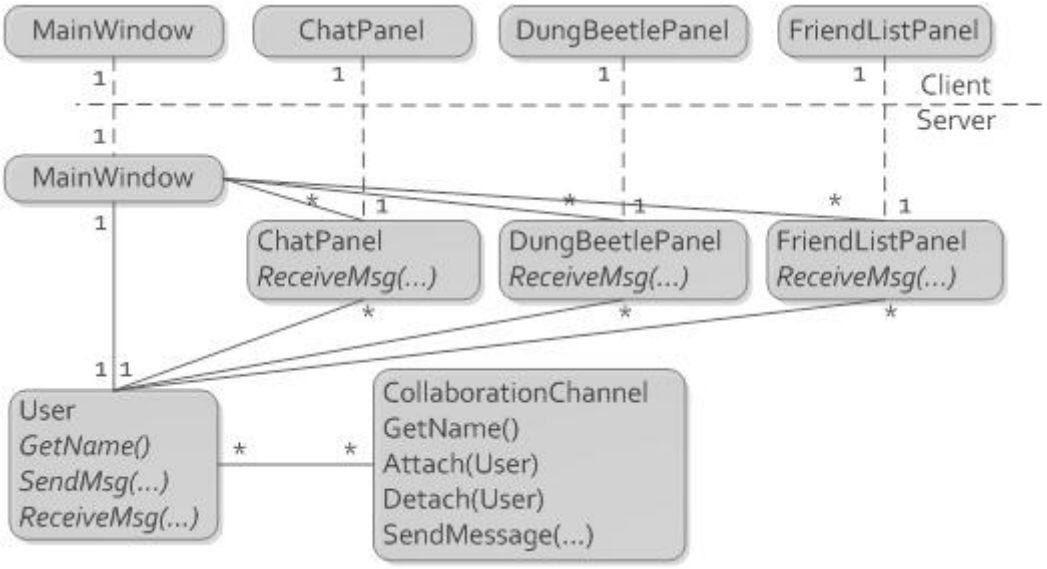

Figure 12. Class structure of real-time collaboration in Vaadin framework

On the server-side, CollaborationChannel and synchronized Java methods are used for modifying the state of the game. Therefore, unlike in the previous Dung Beetle versions, there is no risk of conflicting states or synchronization problems as only the single instance of the state exists, on the server. The state is then only mirrored to panels on the client-side. The downside of the approach is the network latency that affects the game. The round trip to the server and back always takes place before the view can be modified or new action taken. Hence, depending on the network configuration, the game may not be as smooth as in the Lively Kernel and in MUPE frameworks, where a player can continuously take actions without waiting for the server and the view updates: notifications of each action are sent to the server but we do not need to wait for any confirmations. In the Vaadin version, the view simply does not change until the roundtrip to the server has completed.

The sequence diagram in Figure 13 presents some typical game actions. At first, the server randomizes a new dung location and updates the state of the game. Then the server sends message to clients and views are updated accordingly. Then Player1 moves the Beetle. The notification is sent to the Server that updates the state of the application accordingly. As the state of the game is again changed, the server sends messages to both clients that again use the received description for updating the views.

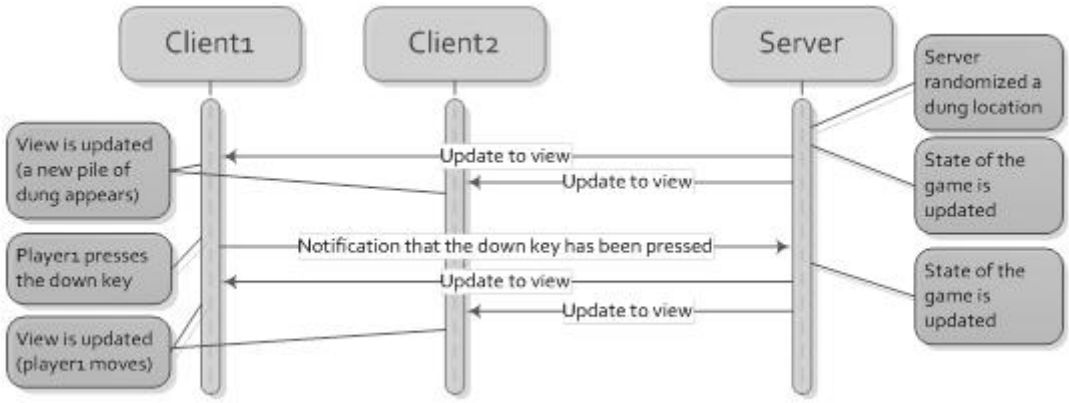

Figure 13. Sequence diagram of Dung Beetle for Vaadin

3.3.4. Mainstream Framework – Node.js

Many third party libraries that extend Node.js exists. For implementing real-time multi-user applications NowJS [NowJS13] or MeteorJS [Meteor13] can be used. In [VII] we perform some hands-on experiencing with NowJS: in brief, the NowJS introduces a new namespace called *now*, which can be used for calling functions and sharing synchronized variables over a WebSocket [Hickson11]. With MeteorJS the applications are also implemented using JavaScript and it contains methods for synchronizing the clients and server-side. Furthermore, MeteorJS contains some additional features like automatic updates of the visible page if an HTML template or data in the database changes and methods for compensating networking latency.

No Dung Beetle implementation for Node.js has been implemented, but similar conventions as in our MUPE implementation could be used, and by utilizing NowJS library, the task should not be too difficult. It is important to notice that Node.js and NowJS allows several different solutions, but in the following we will describe only one of the possibilities. For client-side, we could implement a JavaScript application that would be able to store and show the state of the game and process key events. Server-side would relay actions and randomize the locations of new dung piles. The sequence diagram in Figure 7 can be also used for our NowJS-based game. In Figure 7, the server first randomizes the locations of dung pile and uses synchronized *now* namespace for calling client-side functions. As the players move, the movement handler function from the server is called over the *now* namespace. Furthermore, the namespace is used for notifying the other player of the new location.

3.4. Summary

In this Chapter, we introduced some commonly known examples of multi-user and real-time multi-user web applications. Furthermore, we discussed the current trends and presented some different categories of multi-user web applications. From our perspective, the most interesting of categories are editors and games. This is because

many of the online editors already support real-time collaboration and in games there is potential for impressive real-time multi-player demonstrations.

As an example, we presented a real-time two-player game called Dung Beetle and described observations collected during the implementations. Dung Beetle is implemented for three different frameworks that are MUPE, Vaadin and the Lively Kernel. We used these example frameworks to explain how multi-user applications are implemented and discussed about communication and synchronizing data between the clients. We also discussed how the game could have been implemented using Node.js. Table 3 summarizes, for each, the responsibilities of the client and the server, discusses on how the communication between clients and the server can be enabled, and makes some remarks about challenges and problems.

4. DEVELOPING MULTI-USER REAL-TIME WEB APPLICATIONS

This chapter concentrates how to implement real-time multi-user web applications. The current trend in web applications is to get rid of plugins like Flash and Silverlight and use JavaScript instead [Lawrel12]. However, many web applications like Facebook, Twitter, and weather forecasts, offer custom clients – commonly called apps – for mobile devices these exist to provide improved user experience. For a developer, the most fundamental difference between these native applications, custom clients, and web browsers is that the technology stack of the browsers is restricted to technologies described in the previous chapters. By implementing a native client, it is possible to select the most suitable standards and techniques for that particular case. However, with the browser approach, the potential number of users is much higher since the application is available for wider variety of devices and no installations are needed.

4.1. The Guideline for Designing the System

Before implementing a real-time multi-user web application for a new framework, three obstacles have to be tackled. First the bidirectional communication between the client and the server has to somehow be enabled. Second, a solution for the grouping of the users and for sending the messages inside the server has to be found. Finally, it has to be designed how the shared application state is synchronized between the clients. After these tasks, the actual implementation of the application should not be much more difficult than implementing a normal web application. It should be noted that some frameworks or third party libraries may offer ready-made solutions for these problems.

For multi-user real-time web application the bidirectional communication is an essential feature as the system needs to be able to push messages from server to client. Therefore, the logical first step in application development is to study the alternatives on how the server push can be enabled for the selected web application framework. In Section 4.2, some different alternatives are described.

Many of the current web application frameworks allow enabling bidirectional communication, but in many web application frameworks, the developer needs to select and implement the abstractions or uses third party libraries. In Sections 4.3, we describe how the publish/subscribe pattern can be used at the server-side for creating groups and communication channels. Publish/subscribe pattern is not the only possible abstraction for this, but it should be possible to use it for the most of the purposes and frameworks.

Often the multi-user real-time application has a state that needs to be synchronized between the clients. For example, in Dung Beetle game, the state of the game is synchronized. Depending on the thickness of the client, the methods for synchronized the state may vary. In Section 4.4 some methods for synchronizing the states are discussed.

4.2. Alternatives for Communication

As already discussed, the current web technologies were originally designed for viewing static documents, not for real-time multi-user web applications. The HTTP protocol is fundamentally asymmetric: client initiates the communication by sending HTTP request to the server that respond with an HTTP reply. Pull-based communication is a valid approach for accessing static interlinked documents, but it is not a good starting point for implementing multi-user web applications as there might be a need for incremental updates flowing from the server without client initiative.

However, in the default case, in order to deliver a message, the server needs to wait until a request is made and then piggyback message with the response. The crucial missing operation from HTTP is the server initiated communication, often referred to as "server push". In collaborative application like chat the need for this is obvious, but in news sites, weather forecasts, or stock market pages, the data at the server-side updates constantly, and in some cases it would be a required feature to be able to push new data to the active users as soon as possible.

4.2.1. Ajax with Polling

In a browser environment, the client-side application is implemented using JavaScript or plugin-based techniques. For asynchronous non-blocking communication, a JavaScript application can use XMLHttpRequests [Kester07]. [Garett05] coined this kind of a communication framework with term Ajax. Today, Ajax is an important part of web applications, and many JavaScript libraries offer simple interfaces for Ajax communication.

In HTTP based-communication, the server is not able to send notifications to the client, but the server is able to queue notifications and send them later together with further responses. The queue-based approach is used in the Vaadin framework. Queuing is a good addition to a request-response-based system, but it does not solve the notification problem, because if no requests are made the events just accumulate on the server. Using queues forces the client to poll the server; how often will depend on the case.

In a simple browser-based chat application, the user first writes a message in the field, and then pushes a button for submitting the message to the server, and finally the message is stored as a new row to messages table in the database. For acquiring a new

message, the client regularly initiates new requests. Depending on the need, polling can be done more or less often, but the trade-off between latency and efficiency exists: if done too often, most of the responses do not contain any payload data and bandwidth is wasted; if done too rarely, the latency grows.

4.2.2. Comet

Server push solves the polling problem as the server is able to initiate the messaging when needed. Many solutions for emulating server push exist. Most of them fall under Comet, an umbrella term coined by Russell for the low latency server initiated data transfer [Russel06]. Comet is not an explicit set of technologies but a term to describe server push data functionality, and it can be implemented in several different ways. Commonly used approaches rely on long-held HTTP requests that allow the server to push data without explicit request. To be more accurate, an explicit request is made, but the server delays the response until it has something meaningful to respond with. The downside of Comet is that if no response has been received in the timeframe defined by the browser, the connection timeouts, and thus the connection has to be periodically renewed [McCarthy08, Mueller11, Resig06].

4.2.3. WebSocket

The developers of real-time multi-user web applications would most often like to find a working effective standard answer for enabling bidirectional communication. In connected devices and custom client systems the traditional way to do this is to use socket connections or libraries build on top of them. An accompanying to the HTML5 specification [Hickson08] defines a socket standard for browsers. WebSocket [Hickson11] is a bidirectional, full-duplex communication channel over a TCP socket. Both the server and client can use WebSocket for sending data at any time or even at the same time. The data can be sent without HTTP headers, and this reduces bandwidth usage dramatically in case of small incremental updates. However, the approach is not universally available among the browsers, and in some browsers the feature has to be manually enabled because of security concerns [Roth10].

[Gutwin11] presents some measured performance differences between Comet techniques and WebSockets. In their wide area network configuration, the round-trip latencies between client and server for Comet was little less than 200ms and for WebSockets less than 100ms. In another test, they tested how many packages of 500 bytes it was possible to send per second. By using XMLHttpRequests, it was possible to send around 20 packages from client, but by using WebSockets the rate was over 3000 packages. From server to client the rate was 20 notification packages by using Comet and over 5000 by using WebSockets. Even if the differences seem huge and the numbers would vary greatly depending on the network setting, both approaches are probably good enough for many real-life cases and networking is not the main limitation in delivering real-time multi-user applications in the browser. [Fettweis12] defines four types of physiological real-time constraints: muscular, audio, visual, and

tactile. Based on the study it can be summarized that WebSockets are enough for the first two. Visual and tactile constraints seem to be something that we cannot achieve with the state-of-the-art web technologies.

4.2.4. WebRTC

Another free standardized open source API for real time communication is WebRTC that aims at real-time communication without plugins and is already available for Google Chrome [Bregkvist12]. WebRTC brings peer-to-peer (p2p) real-time communication to browsers and, thus, fills one missing key piece of the platform. WebRTC is a part of the Chrome project, but the goal is to make it a cross-industry solution. WebRTC offers MediaChannel and DataChannel for easily streaming media and data between JavaScript based client-side applications. The web server is still needed, but only for serving the content and enabling users to discover each other. With MediaChannel it is possible to implement communicating web applications like Skype for the browser, and the DataChannel can be used in many other domains like in gaming, remote desktop applications, real-time text chat, and file transfer. Compared with other solutions, the advantage of WebRTC is that the messages travel directly between the clients. Therefore, the lag is reduced as the messages are directly routed between the clients. In addition to JavaScript API, a C++ counterpart for native applications is planned [Dutton13]. In thick client applications, WebRTC seems a very appealing approach, in particular in cases where the server has no need to monitor or control the messaging between the clients.

4.2.5. Other Standards and Protocols

Several standards and protocols for implementing bidirectional communication exist. PubSubHubbub [Winn09], extends Atom [Nottingham05] and RSS feed [Winer03] protocols and provides near-instant notifications for the new feed items. Basically news sites and blogs could enable real-time features by using PubSubHubbub approach. Furthermore, application types like messaging and chatting can be implemented using PubSubHubbub.

In a native client environment XMPP [Saint-Andre09] is often used for enabling bidirectional communication in applications like file transfer, games, instant messaging, or voice over IP. It is an open technology for real-time communication, using XML. It is standardized and proven technology with over fifteen years of development and millions of users. With the help of JavaScript libraries, XMPP can also be used for enabling communication between a browser and a server.

Sometimes an additional notification channel is used for signaling the client that new data has become available for downloading. In modern web applications WebSockets could be used as a notification channel. Notifications can be used for enabling bidirectional communication in a RESTful system. In RESTful systems, the data is presented as resources with unique URIs. For example, in a RESTful chat application,

each chat message could be exposed as a resource with a URI. The approach could, of course be based on polling but by polling we could easily end up polling several different URIs constantly. This could also be averted, for example, by exposing a resource called new notifications and by polling that.

It is also suggested that in a RESTful architecture a new HTTP method called WATCH alongside GET [Khare04] could be added. Instead of getting a representation of a resource, WATCH would make a long-running request and changes to the resource would cause a NOTIFY to the client. This would be a very RESTful answer to bidirectional communication, but in practice it is a significant implementation challenge to introduce a new HTTP method across the public Internet.

4.3. Abstracting the Communication

Comet and WebSockets can be seen as an answer to enabling the bidirectional communication between the server and client. For enabling direct communication between two clients WebRTC can be used. However, to make communication in general simpler in a framework for multi-user applications, a layer of abstraction needs to be built on top of them. WebSocket simply enables passing and receiving messages. [Eugster03] mentions Message Passing as the ancestor of the distributed communication and suggests that even if complex interaction schemes can be built on top of it, it would help developers if the communication pattern was less primitive. Furthermore, they list several alternative communication paradigms that could be built on top of the existing primitives like Remote Procedure Call (RPC), Notifications, Distributed Shared Memory (DSM), Message Queuing, and Publish/Subscribe model.

Most web application frameworks offer the databases as a standard solution for enabling the communication between the users. In applications like Flickr or Twitter, users can upload new data (pictures or text) to the system and to see the new content, the other users reload the page. Hoverer, along with the databases, we need another pull-based layer for communication. The database and server may be located on different computers and, therefore, network lag can again be an issue. Furthermore, as with HTTPRequests, we do not want to end up constantly querying the database for changes. To solve the problem, many database systems offer database triggers. The lowest common nominator between the systems seems to be a possibility to write a change log and then poll the log to find updates. Depending on the database engine there may be more advanced alternatives. Some database engines are even able to make an external function call when triggering takes place.

Luckily, in many frameworks the direct function-based communication between the users is allowed. In particular, in Vaadin it is easy to implement a communication class that enables interactions between the users. For example, the singleton pattern [Geary03] or static classes can be used. With a shared class or object it is easy to implement a capability for creating new channels and enable subscribing to channels

[Vaadin13]. This does not mean that databases have become obsolete. For example, in two-player Tetris where the players see each other's actions, the game actions could be function calls, and the high scores could be stored into the database.

In publish/subscribe pattern (Figure 14) users subscribe and unsubscribe to specific communication channels [Eugster03]. A publisher sends messages to a channel that forwards them to the subscribed users. [Hall96] provides an example called Corona in which publish/subscribe paradigm is used for enabling collaboration. [Huang04] studies the model in a mobile environment. The findings presented in the paper support the assumption that the model would also suit web applications. Publish/subscribe is able to quickly adapt in a dynamic environment with frequent connections and disconnections. The architecture can also be extended to use a message broker if we need to deliver messages to the users that are not currently online.

Figure 14. Publish/Subscribe pattern [Eugster03]

4.4. Shared Data between the Clients

A naïve approach to the consistency of a distributed system would be to hope that the distributed data stays synchronized. However, this is difficult to ensure as latencies differ and, therefore, data storages see updates at different time and in different order. The consistency models are used for defining how to synchronize data. The list of different consistency models is long, but for example causal, eventual, and sequential consistency models are commonly used. In a system with sequential consistency [Lamport79], every node of the systems sees the operations in the same order. In eventual consistency [Vogels09], the order does not matter, but given a sufficiently long period of time without changes, the updates are propagated throughout the system and the replicas will be synchronized. In causal consistency [Ahamad93], the operations that are causally related are replicated to every node in the same order. The operations that are not causally related may be seen in different order by different nodes.

For example, in the collaborative text editor, the straightforward implementation would be just multicast messages like "add x to location 5" or "remove location 5". It turns out that things are more complicated because of network lag. If users add "x" and "y" simultaneously to location 5, then how it is decide if "xy" or "yx" is the correct outcome? What if one user deletes the whole text and another tries to do "add x to location 5". Furthermore, how to ensure that all the clients have made the similar

decision and how to decide which client hold the right version of data? In our work, eventual consistency is used. In the case of a text editor, the documents between the clients may vary while editing, but eventually all the distributed versions of the document are synchronized. Basically, in the example above, the system just somehow decides which of the outcomes is the right one and replaces the other outcome.

Sometimes, estimates for the future data are needed and the real data has to be somehow merged to the prediction. For example, in real-time multiplayer games like first person shooters and MMORPGs predictive algorithms make the movements and actions of the players smoother [Nokia03]. Nevertheless, with a high lag and a poor algorithm this can cause characters colliding with the obstacles and other random effects.

In [VI] we ensured eventual consistency in a collaborative code editor by storing the up-to-date data on the server and then mirroring it to the clients using algorithm presented in [Fraser09]. While using this approach, the client versions cannot drift apart because they just mirror common data. If application is fast paced, there may be problems on how the mirrored version on the client is adjusted to the real version on the server. If a conflict occurs, the mirrored version on a client can always be replaced by the version on the server, but this should be done smoothly and in user friendly way. In MUPE and the Lively Kernel implementations of Dung Beetle, the thick client approach was used. Hence the game was smooth to play, but it also caused some special situations in cases when Beetles interacted with each other's by biting or by simultaneously pushing the same pile of dung. In the Vaadin implementation, the data is stored at the server and, therefore, no problems with the synchronization of situations existed, but in case of slow network the game was not so smooth to play.

4.5. Summary

In this chapter, we concentrated on the key problems in enabling a real-time multi-user approach for web applications and discussed how to implement multi-user real-time web application. In the most current frameworks it is possible to bypass the problems, but the frameworks do not offer any out-of-the-box solutions. MUPE was a multi-user application framework used in J2ME-based mobile devices and it contained many helpful features that could be transferred to the browser world. Our opinion is that there are three separate problems: how to enable the bidirectional communication between the client and server, how to build helpful abstraction on top of the communication, and how to share data between the clients.

No single best solution for the problem exists, but Table 4 describes a possible solution.

Table 4. Solutions for problems in real-time multi-user web applications

Problem	Solution
Bidirectional communication between the client and the server	WebSockets enable bidirectional communication and decrease the lag and data overhead when compared with other alternatives available. However, it may be necessary to include Comet fallback alternative to ensure compatibility.
Abstracting the communication between the clients	To form user groups and for abstracting the communication inside the server, for example, the publish/subscribe pattern can be used.
Shared data between the clients	For enabling an easier development process, we suggest storing application logic and state at the server-side. Therefore, only one state exists and can be mirrored to each of the clients. However, certain benefits can be gained by thickening the client-side, but then the synchronization of the states is more difficult

5. RELATED RESEARCH

All the separate aspects and fields that we addressed in this dissertation are widely studied, but to our best knowledge this is among the first attempts to combine all of them. In gaming, multi-player real-time native games have been around almost as long as consumers have been able to do networking. Furthermore, lots of research papers have been written concerning web applications and multi-user web applications, but it seems that in this stage they still mostly concentrate on example custom-built web applications and more rarely on custom-made multi-user web applications. Examples of real-time multi-user web applications are even less known, although some examples do exist. Furthermore, we were also able to find some solutions that aim for more generic frameworks for implementing real-time multi-user applications. In the following some approaches that touch our research are presented.

5.1. Real-Time Multi-User Applications

Collaborative editing is probably the most commonly used example of real-time multi-user applications. In the early ages of computing, the term teamware was used for describing the collaborative systems and WYSIWIS (What You See Is What I See) was used to describe simultaneous editing. In [Magnusson93] a native system built on top of the Mjølner Orm environment [Magnusson90] is presented. The system follows the client-server architecture, and it supports collaborative text and code editing. In a later article [Minör93] authors describe what kind of situations collaborative editing is needed in and how it should be implemented. The paper divides synchronous and asynchronous collaborative editing into two categories and explains the situations where they can be used. Furthermore, the paper presents the third method: the semi-synchronous editing that is a model for combining the good features of both extremes. The semi-synchronous editor presented relies on three concepts: 1) hierarchical organization that uses a specific grammar for partitioning the document; 2) fine-grained version control that maintains the versions; and 3) active diffs that provide group awareness by showing the differences between the versions.

GroupKit introduced in [Roseman96] is a groupware toolkit that enables developers to build synchronous and distributed native applications. The infrastructure of GroupKit automatically takes care of some tasks and offer groupware programming abstractions for developers. The goal of the toolkit was to reduce the level of the difficulty of implementing a multi-user to only slightly more difficult than that of a single-user application. To reduce the difficulty, the authors offered abstractions, widgets, and primitives for remote procedure calls, sharing data, session management, registration

process, and conferencing events, all of which can be extended for the further functionalities. For remote messaging, one-to-one (unicast) and one-to-many (multicast) communication was offered. The paper explains how applications were implemented using GroupKit and what a GroupKit program looked like. The article also compares GroupKit with the other groupware toolkits of that age.

[Ellis98] discusses operational transformation in real-time group editors as well as related issues, algorithms, and achievements. Operational transformations have become a solution often selected for enabling communication as they can widely be used in different application domains. In a nutshell, operational transformation algorithms are a method for maintaining the consistency in real-time multi-user editors. As a minimal example, a text editor can be implemented using two operations: `O1 = Insert[0, "x"]` that inserts character `"x"` in position `"0"`, and `O2 = Delete[2, "c"]` that deletes the character `"c"` in position `"2"`. In a more complex text editor, additional operators like `Move` and `Replace` could come in handy. In addition to the operations, the rules for resolving conflicts from concurrent operations are needed.

Plugin-based application frameworks for the web browser often offer some support for real-time multi-user applications. For example, Flash contains XMLSockets [Adobe13c] that are similar to WebSockets. Disney has offered a huge amount of Flash-based games, and some of them even have multi-player mode. Furthermore, they also offer examples and guidelines on how to enable communication between the players [Lee04]. However, to our best knowledge, no complete frameworks for making real-time multi-user applications exist for Flash. Furthermore, Flash is a component that user has to separately install and update, and many device manufacturers have decided not to support Flash-based content [Lawler12]. Other plugin-based approaches like for example Microsoft Silverlight seem to offer the same kind of feature set, and also share the same limitations as Flash.

MUPE was not the only custom client framework for implementing multi-user applications for Mobile devices. WidSets by Nokia offered some similar features. After initial WidSets client installation the selection of widgets could be browsed and used. For WidSets, an SDK containing an emulator for creating widgets were offered. In some simple tasks, like in simple UIs or in showing RSS feeds, WidSets was an even more advanced framework than MUPE was. In addition, the documentation of the framework promised that it could be used for creating multi-user applications although this was not an essential part of the system. Unfortunately, the testing of these features was difficult as limited number of code examples were available and only one instance of the emulator could be launched at a time. However, while MUPE was never a commercial success, WidSets was able to attract a regular user base and small number of third party programmers to the community [Nokia13].

[Sun06] offers a method for converting existing single-user native applications to collaborative ones. The invention reused generic operation transformations and a

collaboration engine by creating an application-specific adapter for mapping special operations to a set of primitive operations. In this way, a generic collaboration framework can be used without changing the original application. However, as a down-side, a collaborative adapter has to be implemented for each converted application. As an example of their approach, they have created adapters for MS Word and PowerPoint and extended them to multi-user applications that they have named CoWord and CoPowerPoint [Sun06]. Those applications are promised to allow users to be able to collaboratively view and edit documents over the Internet.

Simultaneous collaboration can be used in a wide variety of fields. [Kurki10] describes the components for building a fiction web service described in [Hypén10]. The system can for example, be used for adding and searching information in Finnish libraries.

In [Wilde11] a system for exposing user interface elements over a RESTful HTTP interface is described. Each UI element has its own URI, and specific monitor resource is used for sending update notifications to the user agents. The system uses an extension of the QT framework [Digia13] and Remote-MVC pattern [Stirbu10] for presenting the UI elements. The described approach supports multi-displays and multi-users. As an example application, a Texas Hold'em game with the server running on a Ubuntu laptop, a TV used as a public display, and Nokia N900 smartphones as private gaming devices is presented. While playing, the public information is presented on the TV screen and private information on the smartphones. The approach is promising and even if the QT framework is used as a presenting layer, nothing seems to prevent developers from mimicking the same infrastructure and using the browser-based UI-libraries instead.

Some commercial collaborative frameworks have emerged recently. One of those is beWeeVee [beWeeVee13] that uses operational transformations. It can be used for implementing desktop applications with .NET or web applications on Microsoft Silverlight. Data structure changes are transformed by beWeeVee to canonical operations that can be propagated to remote sites over peer-to-peer or client-server architectures. Furthermore, beWeeVee enables features such as unlimited undo and change playback.

5.2. Real-Time Multi-User Web Applications

There is a variety of good real-time multi-user web application examples available. [VII] list some game examples, [VI] some collaborative editors, and in Section 3.2 some additional examples. In addition to these there are plenty of scientific publications that concentrate on some fields of the area; in following, we list a few.

Real-time collaboration can be used to enable virtual laboratories that are accessed using browser [Jara09]. In these laboratories students can train and learn laboratory tasks through the Internet and a teacher can guide them remotely in real-time.

Some methods for tuning traditional web applications or pages towards the multi-user paradigm exist. For example, [You04] introduces the People Awareness Engine that can be added to a normal single-user web application or page and is then used for real-time communication with other users. People Awareness Engine does not make the original page collaborative but it makes the people at the same site visible and enables interaction.

Furthermore, [Lowet09] proposes a method for co-browsing single-user web applications. At least in theory, the approach supports all web pages, but the model of collaboration is limiting for multi-user applications, since all of the browsers share exactly the same view and state. Earlier implementations have made it possible to browse static pages, but because of the dynamic nature of web applications, this approach works differently. [Lowet09] presents two approaches. The first one synchronizes the output of the JavaScript engine by synchronizing the changes made on DOM level. The second one synchronizes the input, like the UI events and incoming data for the JavaScript engine, and thus indirectly also synchronizes the output and the DOM tree.

Another example working at DOM level is presented in [Heinrich12] that proposes a method for converting conventional web applications into multi-user web applications by utilizing a generic collaboration engine and operational transformations to synchronize DOM between the users. The method is named Generic Collaborative Infrastructure (GCI). DOM mutation events are the same for all applications and, therefore sending the DOM events is a better way to generalize application behavior than sending of application specific events. As an example, single-user text and graphic editors are converted to collaborative ones. The approach may be valid for synchronizing the views between the users, but often it is desirable to show different information in different clients even when collaborating, and in such situations, the synchronization of DOM is not the right alternative.

Heinrich et al. have extended their work in several articles. In [Heinrich13a] they analyze which kind of web applications are suitable for transforming from single-user to multi-user using GCI. [Heinrich13b] makes an observation that there are two different approaches for implementing collaborative web applications: (1) Concurrency control libraries and (2) transformation approaches that are capable of converting single-user applications to multi-user application. Multiple examples of both categories are explained. Furthermore, they came up with a new approach that uses source code annotations that would be interpreted at runtime by the collaboration engine. They claim that by using annotations, multi-user web applications are easier and faster to implement. Furthermore, in [Heinrich13c] a reusable set of workspace awareness widgets is introduced. The library can be used for enhancing collaborative applications with features that allow user to be aware of other users and their actions.

5.3. Web Frameworks

In Chapter 4, technical selections for implementing real-time multi-user web applications are elaborated. While the real-time multi-user web applications and frameworks are not a widely studied field, many of the enabling facilities are more deeply researched. Next, we present some interesting highlights of the research that is closely related to our work.

In Section 4.2, we presented up-to-date approaches like Comet and WebSockets for enabling the server push. At the 1990s a company called PointCast implemented the push-based PointCast Network that was first used for displaying live news and information in screensavers [Ramakrishnan98], and during the browser wars, Netscape and Microsoft integrated the technology into their browsers. However, time was not yet right for the technology. The main argument against the PointCast Network was that it used too much bandwidth for the internet infrastructure at the time. Therefore, many networks banned the technology, and it was more or less replaced by pull-based RSS feeds.

For abstracting the communication, different alternatives exist. In Section 4.3, publish/subscribe abstraction was presented, but in multi-user web application approaches like Remote Procedure Call (RPC), Notifications, Distributed Shared Memory (DSM), or Message Queuing could be used as well. In systems utilizing Remote Procedure Call [White75], it is possible to invoke functions over the network. From a developer point of view, remote procedure calls look exactly like local procedure calls: parameters can be passed and return value used. However, it should be noted that even if local procedure calls are cheap, the RPCs are not, because they require networking. In Distributed shared memory [Hennessy07], the physically separated memories can be addressed as one address space. This shared memory can then be used for messaging between different components in a distributed system. Message Queuing [Dickman98] provides an asynchronous communication protocol where messages are placed in queues to be retrieved latter.

Section 4.4 discusses synchronizing the data between the users. However, sometimes it does not make sense to synchronize or even allow all the edits. For example, one meaningful edit may consist of several smaller increments. [Jiang08] discusses on how to implement constraints on the collaborative graphic design system so that collaborators are not able to break semantic consistency. For example, if one of the users draws a face then he probably does not want other users to move an eye to a wrong place. The same kind of problem is presented and solved in a real-time collaborative Web IDE Collabode [Goldman11]. In publication, they have concentrated on an algorithm that only synchronizes error-free edits to collaborating users. The algorithm is useful and we have adopted it with some modifications to CoRED that is presented in [VI].

Open Cooperative Web Framework (OpenCoWeb) [OpenCoWeb13] is an open source project aiming for collaborative web applications. It sends notifications of user changes, resolves the conflicting changes, and uses operational transformations for converging the data of the applications. In addition, the framework offers the so called Coweb client and server for enabling the collaboration. Based on the JavaScript interface, the framework seems quite a low level system, which allows implementing a wide variety of application types on top of it. As a downside, this also makes it more difficult to use. For example, content is synchronized using `CollabInterface.sendSync(name, value)`, in which `name` identifies the property changed and `value` contains new property in JSON format. The rest of the framework seems to work on the same abstraction level. As a simple example, the authors offer the implementation of cooperative shopping list that can be filled by several authors simultaneously. The simple example consists of half a dozen files.

[West12] discuss the meaning of trust in collaborative web applications and explain certain characteristics that make them advantageous to attackers. Authors are not concentrating on real-time collaborative applications, but in many cases the security concerns and methods for avoiding them can be applied to real-time applications.

6. CONCLUSION

The Web is becoming a platform for web applications, and the variety of applications is constantly growing. When the users of web applications are connected to a single server it seems reasonable to assume that they could somehow interact with each other. Along with Web 2.0 some collaborative features have emerged to the Web and users have adopted them. However, it should be noted that new collaborative features are mostly slow, asynchronous collaboration such as updating wiki pages, sending messages, posting blog entries, or adding comments. These are not the features that we are searching for. We want to implement multi-user applications where users can interact with each other in real-time. Well-known examples like chats and collaborative text editors like Google Docs exist, but the standards for implementing the feature are still missing.

The main contributions of this thesis are: pinpointing the main difficulties in the implementation of real-time multi-user web applications, and the finding of the solutions. As a result of the research, we came to the conclusion that the technological stack has not changed much since when the Web just contained interlinked hypertext documents. Web applications are still constructed using the same building blocks as they were used fifteen years ago. For client-side, these building blocks are HTTP communication, HTML markup, CSS style sheet, and JavaScript scripting language. For server-side, a variety of alternatives exists, but solutions mostly build on the same conventions. The gap between client-side and server-side and another between server-side and the database are by default pull-based; in the first case pull operations are called request and in the second queries. Normally, the server cannot freely push new data to the client, nor can the database inform the server about new data being stored.

The problem described above is technical in its nature, but it affects also coding conventions and the architecture of web applications. We considered an alternative structure for the client-server based architecture by studying MUPE (Multi-User Publishing Environment) by Nokia Research Center and came to the conclusion that many of the features that were available in MUPE could be transferred to web applications just by implementing the server differently. We also made a suggestion that the database should not be used for enabling real-time communication. The database can be used in applications, but it should be used for storing the data, not for performing real-time interactions. As an example, in a web-based two-player game, the interactions between the players do not need to be stored in the database, but the high scores or other permanent outcomes definitely should.

Finally, we presented problems, discussion, and solutions concerning real-time multi-user web applications. We were able to find three separate problems: how to enable the bidirectional communication between the client and server, how to build helpful abstraction on top of the communication, and how to solve concurrency and consistency problems.

6.1. Research Questions Revisited

The research questions presented in Chapter 1 are revisited in the following. The main question was divided to several sub-questions and before the main question, the sub-questions are answered:

Q1: *How to enable communication?*
There are several valid alternatives for implementing communication between distributed components in web-based multi-user applications. For slow paced applications, it may be the best alternative to select XMLHttpRequests and polling since the approach is easy to implement and not so resource consuming if done infrequently. In real-time multi-user web applications, the selected technology depends on the parties that are communicating. If all communication travels through the server, WebSockets are the fast and obvious alternative. For enabling the direct messaging between the clients, WebRTC can be used. For supporting older browsers, Comet communication may be necessary to implement as a fallback. Enabling the communication is discussed with more details in Section 4.2.

Q2: *How to enable multicasting?*
The content that arrives to the server can be divided into two separate parts. The first part is content that has a short lifespan and becomes meaningless after it has been relayed to the selected group of users. Groups can be managed, for example, by using the Publish/Subscribe pattern. More information about the internal structure of the server is offered in Section 4.3. After the data has been relayed to selected group of users, it can be consumed, and incorporated to the application state. How the incorporation is done depends on the partitioning of the application and is discussed with more detail in answer to Q3. As usual, the content with longer a lifespan can be permanently stored in files or database. In gaming, these contain permanent outcomes, such as the high score points, and, in a real-time text editor, the complete text.

Q3: *How should the application be partitioned between the client and the server?*
No single answer to the problem of partitioning exists. In a thin client system, multi-user web applications are easier to implement as the state of the application is stored and synchronized on server-side, and only the user interface is mirrored to the clients. However, the latencies in communication cause the server-centric approach to be less responsive than thick client applications. The partitioning choice must be based on the selected framework and the application under development. More detailed answers to this question are offered in Section 4.4.

Q4: *How do development tools and frameworks support the development process?*
The web application frameworks seldom offer help for implementing real-time multi-user web applications. However, if they would offer help, then it should be easier for the development tools to facilitate the implementation of real-time multi-user applications. The tools can, for example, help developers to follow the conventions and offer code completions and other supporting features. If features are not incorporated to a framework, then the best that development tools can offer are libraries, extensions, code snippets, and guidelines. In Chapter 3, we present three web application frameworks and explain how they support the implementation of real-time multi-user applications. In Chapter 5, the scope is broadened and some alternative approaches are presented.

Our main research question that we presented in the first Chapter was:
How to easily develop web applications with real-time multi-user features?

The current technology stack of web applications enables developing multi-user real-time web applications. By nature, these have more complicated requirements and are more difficult to implement than most web applications. However, sophisticated development tools and frameworks may facilitate the multi-user paradigm and leverage the implementation to be just slightly more difficult. Currently, no mainstream framework offers a complete solution for easy development of real-time multi-user web applications and therefore framework can be selected based on other needs and experiences. However, some frameworks already offer support some features and are, therefore, better starting point for implementing real-time multi-user web applications. For enabling the easy implementation of multi-user applications, abstractions on top of the communication layer are needed.

No solution that fits all the frameworks exists and, therefore, the issues need to be tackled one by one, so, the sub-questions above are needed to give the detailed answer. [IV], [VI], and [VII] present real-time multi-user example implementations for the Lively Kernel, Vaadin and Node.js. Section 4.1 presents some guidelines for extending web application frameworks to support multi-user real-time features.

6.2. Future Work

This work concentrates on describing the different approaches for real-time multi-user features and it could be extended to several directions, some of which are described next.

6.2.1. Real-Time Multi-User Web Application Framework

Earlier we made an observation that custom client framework MUPE had a plethora of helpful features that could be transferred to web applications. Vaadin is a web application framework that has certain quality in the way how it enables easy web

application development. One obvious future work would be to implement an application framework that combines the best parts of MUPE and Vaadin to a real-time multi-user web application framework. Its superiority over conventional frameworks could be proven by organized coding sessions where inexperienced developers would implement real-time multi-user web applications under the supervision and guidance of experienced developers using different frameworks. Earlier we have had several intensive courses called code camps on several different themes. Code camp concentrating on multi-user real-time web applications would be a logical continuation.

In our real-time multi-use framework, the user interface and communication between the client and the server could be enabled using Vaadin. This selection leads us to the thin client approach that reduces the level of distribution and, therefore, makes the programming model simpler. However, as a side effect we lose the ability to run client-side logic and, therefore, the responsiveness may not be high enough for all the possible application types.

For enabling the communications between the users, shared objects with synchronized Java methods could be used. A shared object between the users could be implemented using the singleton pattern, a design pattern that restricts the instantiation of a class to one object [Geary03]. Synchronization of Java methods is necessary as several users are accessing to shared objects simultaneously and concurrent requests could otherwise potentially lead to incorrect functionality [Oracle13b].

Singleton based classes could be used for imitating the class structure of MUPE. In simple applications, a single channel could be used for broadcasting all the messages, but often the further division to, say, chat rooms, game instances, or editable documents is needed. This further division can be done using the singleton based Lobby room channel that is able to serve and create communication channels for users to subscribe. In this framework, everything happens at the server-side and the needed parts are mirrored to the clients.

6.2.2. Tool Support

Another interesting branch for future work would be studying tool support. To the best of our knowledge, there are not much tools available for helping development of real-time multi-user applications for a browser. The usual tools like Eclipse or Emacs [Stallman86] can of course be used, but they do not offer much support for developing real-time multi-user features. It seems that instead of trying to create specific tools, the best approach is to include multi-user features in a framework. Without support in the framework, there is not much that can be done just by implementing better tools. Of course, an IDE could offer scripts, code snippets, and additional multi-user libraries. In Java-based systems like Vaadin, Eclipse could offer a lot of help, if multi-user features would be embedded inside the system. Code completion and error checking would work immediately because of the reflective nature of Java.

6.2.3. Navigation Buttons

One interesting theme for future work could to investigate how the navigation buttons of the browser should be supported in a real-time multi-user application. The browser provides additional facilities not available in other systems, such as bookmarking, back and forward buttons, and cloning windows. The facilities can be used for navigating inside a web application and, therefore, the server has no control of the exact client state [Queinnec04]. Most of the time a meaning for those facilities can be invented. When considering the simple drawing application described in Section 2.2, the back button could be used for an undoing edits, forward could then be used as a redo, and bookmarking as a save. Obviously, the meaningful implementation of the buttons totally depends on current application.

In a multi-user environment, the meaningful interpretations for actions associated with navigation buttons are even more difficult to make. Notifications between the users bring a new level of difficulty on top of these issues. Bookmarking could perhaps be used for saving a snapshot of the state, but back and forward button could not be used to go back and forward in the state of the application, since the actions would have an effect on other users, too. In other words, even if we would be able to undo an action by pushing the back button, a notification sent cannot be unsent. In our example applications, these facilities are ignored and the web applications are mostly considered as single page systems where the back button will not undo the last action but backs to the previous page and out from the system.

6.2.4. Context Awareness

Context awareness indicates that a system is somehow linked with changes in the real world, and it is, therefore, possible to react to changes in the environment [Dey00]. Many web applications already benefit from the mobile and context aware aspects. As an example, the Geolocation API can be used in direction services. In Google Maps at “Get Directions” service “My Location” is pre-filled, and therefore in the basic case of navigation, the user only fills in the destination field [Schutzberg09]. In MUPE related research, we have studied context awareness [Suomela04], but in browser-based web applications, the implementation of context awareness is more difficult.

The value of context awareness is more difficult to define on desktop and laptop computers. However, it could be interesting to study how to utilize context awareness in multi-user web applications. As the location API [Popescu10] is already available in the browsers, it would be the easiest starting point. A chat service could suggest Tampere-channel automatically for a person who is in Tampere, Finland. In games or social media, people standing nearby each other could be automatically grouped. In player proximity based.games, the positioning data could be used for interacting with players nearby. Geocaching can also perhaps be seen as a form of position-based multi-user application.

Currently, applications utilizing context awareness could be easily developed for the browser. Many browsers implement W3C Geolocation API for resolving physical location. By using the API, the device can be located based on other information available. Huge databases have been collected for mapping IP, Wi-Fi, and Bluetooth MAC addresses and radio-frequency and GSM/CDMA cell identifiers to physical locations. If a device has a Global Positioning System (GPS) module, then it can be used in web application. The GPS modules of the devices are also used for continuous updating of Geolocation database [Vaughan-Nichols11]. As an example, in the author's office, Google Chrome 22 is able to find the current location with the accuracy of about 10 m.

REFERENCES

[Adobe13a] Adobe Flash Platform, *http://www.adobe.com/flashplatform/* (last accessed 27. March 2013)

[Adobe13b] Adobe Shockwave Player, *http://www.adobe.com/products/shockwaveplayer/* (last accessed 27. March 2013)

[Adobe13c] Reference for the Adobe Flash Platform, XMLSocket class, *http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/XMLSocket.html* (last accessed 12. September 2013)

[Ahamad93] Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. "Causal memory: Definitions, implementation, and programming." Distributed Computing 9, no. 1, pp. 37-49, 1995.

[Andrews00] Gregory R. Andrews. "Foundations of multithreaded, parallel, and distributed programming." University of Arizona, Wesley, USA, 2000.

[Anttonen11a] Matti Anttonen, and Arto Salminen. “Building 3D WebGL Applications.“ Technical report, Tampere University of Technology, p48. Tampere, Finland, 2011.

[Anttonen11b] Matti Anttonen, Arto Salminen, Tommi Mikkonen, and Antero Taivalsaari. "Transforming the web into a real application platform: new technologies, emerging trends and missing pieces." In Proceedings of the 2011 ACM Symposium on Applied Computing, pp.800-807. ACM, 2011.

[Apache13] Apache Software Foundation. Apache Wave. *http://incubator.apache.org/wave/* (last accessed 23. July 2013)

[Apple13] Apple QuickTime player, *http://www.apple.com/quicktime/what-is/* (last accessed 2. September 2013)

[Berners-Lee05] Tim Berners-Lee, Roy Fielding, and Larry Masinter. "Uniform resource identifiers (URI): generic syntax." RFC 3986, p. 61. 1998.

[beWeeVee13] BeWeeVee – Life collaboration framework, *http://www.beweevee.com* (last accessed 2. September 2013)

[Bibeault08] Bear Bibeault, and Yehuda Kats. “jQuery in Action.” Dreamtech Press, p. 452. 2008.

[Bos11] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. "Cascading style sheets level 2 revision 1 (css 2.1) specification." W3C Recommendation, p. 487, 2011.

[Bray06] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. "Extensible Markup Language (XML) 1.1. (Second Edition)" W3C recommendation , p. 45. 2006.

[Bregkvist12] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings, and Anant Narayanan. "WebRTC 1.0: Real-time communication between browsers." Working draft, W3C, 2012.

[Cerf80] V. Cerf and J. Postel. “Mail Transition Plan.” RFC 771, p. 9. 1980.

[Chui12] M. Chui, J. Manyika, J. Bughin, R. Dobbs, C. Roxburgh, H. Sarrazin, G. Sands, and M. Westergren. “The Social Economy: Unlocking Value and Productivity through Social Technologies.” McKinsey Global Institute, p. 170. 2012.

[Colley04] Steve Colley. “Steve Colley's Story of the original Maze: Stories from the Maze War 30 Year Retrospective.” *http://www.digibarn.com/history/04-VCF7-MazeWar/stories/colley.html* (last accessed 27. March 2013)

[Crane05] Dave Crane, Eric Pascarello, and Darren James. “Ajax in action.” Dreamtech Press, p. 650. 2005.

[Crockford06] Douglas Crockford. "The application/json media type for javascript object notation (json)." RFC 4627, p.10. 2006.

[Crockford08] Douglas Crockford. JavaScript: the good parts. Yahoo Press, p. 156. 2008.

[Dey00] Anind K. Dey. "Providing architectural support for building context-aware applications." PhD diss., p. 170. Georgia Institute of Technology, 2000.

[Dickman98] Alan Dickman, and Peter Foreword By-Houston. “Designing applications with MSMQ: message queuing for developers.” Addison-Wesley Longman Publishing Co., Inc., p.370. 1998.

[Digia13]Qt, *http://qt.digia.com/* (last accessed 13. September 2013)

[Dix96] Alan Dix. "Challenges and perspectives for cooperative work on the Web." SIGCHI Bulletin 28, no. 2, pp. 47-49. 1996

[Dojo13] Dojo toolkit, *http://dojotoolkit.org/* (last accessed 2. September 2013)

[DojoX13] DojoX, *http://dojotoolkit.org/reference-guide/1.7/dojox/index.html* (last accessed 2. September 2013)

[Dutton13] Sam Dutton. “Getting started with WebRTC. HTML5 Rocks tutorials”, *http://www.html5rocks.com/en/tutorials/webrtc/basics/* (last accessed 2. September 2013)

[Eclipse13] Eclipse IDE. *http://www.eclipse.org/* (last accessed 12. September 2013)

[Ecma11] Ecma international. "ECMA-262 ECMAScript Language Specification." p. 245. 2011.

[Eckstein98] Robert Eckstein, Marc Loy, and Dave Wood. “Java swing.” O'Reilly & Associates, Inc., p. 1252. 1998.

[Edwards97] W. Keith Edwards, Elizabeth D. Mynatt, Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, and Marvin M. Theimer. "Designing and implementing asynchronous collaborative applications with Bayou." In Proceedings of the 10th annual ACM symposium on User interface software and technology, pp. 119-128, ACM, 1997.

[Ellis89] Clarence A. Ellis and Simon J. Gibbs. "Concurrency control in groupware systems." In ACM SIGMOD Record, vol. 18, no. 2, pp.399-407, ACM, 1989.

[Ellis98] Clarence A. Ellis and C. Sun. "Operational transformation in real-time group editors: issues, algorithms, and achievements." In Proceedings of the ACM conference on Computer supported cooperative work, pp. 59-68, 1998.

[Ellison07] Nicole B. Ellison. "Social network sites: Definition, history, and scholarship." Journal of Computer-Mediated Communication 13, no. 1, pp. 210-230, 2007.

[Engalbart68] Douglas C. Engelbart. "The mother of all demos." 1968. *http://www.youtube.com/watch?v=JfIgzSoTMOs* (last accessed 28. June 2013)

[Eugster03] Patrick T. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. "The many faces of publish/subscribe." ACM Computing Surveys 35, no. 2, pp. 114-131, 2003.

[Express13] Express web application framework for node. *http://expressjs.com/* (last accessed 30. June 2013)

[Fettweis12] G. Fettweis. “A 5G Wireless Communications Vision” Microwave Journal, 2012.

[Fielding99] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. "Hypertext transfer protocol–HTTP/1.1." p.176. 1999.

[Fraser09] Neil Fraser. "Differential synchronization." In Proceedings of the 9th ACM symposium on Document engineering, pp. 13-20, ACM, 2009.

[Freed96] Ned Freed and Nathaniel Borenstein. "Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies." 1996.

[Fried10] Ina Fried, and Lowensohn, Josh. “Google pulls plug on Google Wave.” August 4, 2010. *http://news.cnet.com/8301-13860_3-20012698-56.html* (last accessed 2. September 2013)

[Garett05] Jesse James Garrett. "Ajax: A new approach to web applications." (2005). *http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications* (last accessed 28. June 2013)

[Geary03] David Geary. "Simply Singleton, Navigate the deceptively simple Singleton pattern." JavaWorld How-To-Java, 2003.

[Git13] Git User's Manual. *http://www.kernel.org/pub/software/scm/git/docs/user-manual.html* (last accessed 12. September 2013)

[Goldman11] Max Goldman, Greg Little, and Robert C. Miller. "Real-time collaborative coding in a web IDE." In Proceedings of the 24th annual symposium on User interface software and technology, pp 155-164. ACM, New York, 2011.

[Google10] Google Inc. "Laying the Foundation for a New Google Docs." *http://googleenterprise.blogspot.com/2010/04/laying-foundation-for-new-google-docs.html* (last accessed 2. September 2013)

[Google13a] Google Web Toolkit Overview, *http://developers.google.com/web-toolkit/overview* (last accessed 2. September 2013)

[Google13b] V8 JavaScript Engine, *https://code.google.com/p/v8/* (last accessed 12. September 2013)

[Gray98] J. N. Gray, Raymond A. Lorie, Gianfranco R. Putzolu, and Irving L. Traiger. "Granularity of locks and degrees of consistency in a shared data base." Readings in Database Sys, pp. 175-193. 1998.

[Grudin94] Jonathan Grudin. "Computer-supported cooperative work: History and focus." Computer 27, no. 5, pp. 19-26, 1994.

[Grönroos13] Marko Grönroos. "Book of Vaadin." Vaadin Limited, p. 596. 2013.

[Gutwin11] Carl A. Gutwin, Michael Lippold, and T. C. Graham. "Real-time groupware in the browser: testing the performance of web-based networking." In Proceedings of the conference on Computer Supported Cooperative Work, pp. 167-176, ACM, 2011.

[Hall96] Robert W. Hall, Amit Mathur, Farnam Jahanian, Atul Prakash, and Craig Rassmussen. "Corona: a communication service for scalable, reliable group collaboration systems." In Proceedings of the conference on Computer supported cooperative work, pp.140-149, ACM, 1996.

[Heinrich12] Matthias Heinrich, Franz Lehmann, Thomas Springer, and Martin Gaedke. "Exploiting single-user web applications for shared editing: a generic transformation approach." In Proceedings of the 21st international conference on World Wide Web, pp. 1057-1066, ACM, 2012.

[Heinrich13a] Matthias Heinrich, Franz Lehmann, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. "Analyzing the suitability of web applications for a single-user to multi-user transformation." In Proceedings of the 22nd international conference on World Wide Web companion. International World Wide Web Conferences Steering Committee, pp. 249-252, Republic and Canton of Geneva, Switzerland, 2013.

[Heinrich13b] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. "Exploiting annotations for the rapid development of collaborative web applications." In Proceedings of the 22nd international conference on World Wide Web.

International World Wide Web Conferences Steering Committee, pp. 551-560, Republic and Canton of Geneva, Switzerland, 2013.

[Heinrich13c] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, Philipp Hauer and Martin Gaedke. "GAwI: A Comprehensive Workspace Awareness Library for Collaborative Web Applications". Web Engineering Lecture Notes in Computer Science Volume 7977, pp. 482-485, 2013.

[Hennessy07] John L. Hennessy, and David A. Patterson. “Computer architecture: a quantitative approach.” Morgan Kaufmann, p. 484. 2011.

[Hickson08] Ian Hickson and D. Hyatt. "Html 5: W3c working draft." 2008.

[Hickson11] Ian Hickson. "The websocket api." W3C Working Draft WD-websockets-20110929, 2011.

[Huang04] Yongqiang Huang, and Hector Garcia-Molina. "Publish/subscribe in a mobile environment." Wireless Networks 10, no. 6, pp. 643-652, 2004.

[Hypén11] Kaisa Hypén, and Antti Impivaara. "Read, describe and share! Building an interactive literary web service: an article about Kirjasampo." Collection Building 30.1, pp. 61-67, 2011.

[Ingalls08] Daniel Ingalls, Krzysztof Palacz, Stephen Uhler, Antero Taivalsaari, and Tommi Mikkonen. "The lively kernel a self-supporting system on a web page." In Self-Sustaining Systems. Springer Berlin Heidelberg, pp. 31-50, 2008.

[Ippolito05] Bob Ippolito. "Remote json-jsonp.", 2005. *http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/* (last accessed 28. June 2013)

[Jacobs04] Ian Jacobs and Norman Walsh. "Architecture of the world wide web, volume one." W3C Recommendation, 2004.

[Jara09] Carlos A. Jara, Francisco A. Candelas, Fernando Torres, Sebastian Dormido, Francisco Esquembre, Oscar Reinoso. “Real-time collaboration of virtual laboratories through the Internet” Computers & Education, Volume 52, Issue 1, pp. 126-140. 2009.

[Java13] *Java, http://www.java.com/en/* (last accessed 27. March 2013)

[Jazayeri07] Mehdi Jazayeri. "Some trends in web application development." In Future of Software Engineering, pp. 199-213. IEEE, 2007.

[Jiang08] Bo Jiang; Jiajun Bu; Chun Chen; Bo Wang, "Semantic consistency maintenance in collaborative graphics design systems," 12th International Conference on Computer Supported Cooperative Work in Design, 2008.

[Johansen88] Robert Johansen. “Groupware: Computer support for business teams.” The Free Press, 1988.

[Järvinen00] Pertti Järvinen. "Research questions guiding selection of an appropriate research method." In Proceedings of the 8th European Conference on Information Systems, vol. 3, no. 5.6, pp. 124-131. 2000.

[Kaplan10] Andreas M. Kaplan, Michael Haenlein, “Users of the world, unite! The challenges and opportunities of Social Media”, Business Horizons, Volume 53, Issue 1, pp. 59-68. 2010.

[Kester07] Anne Van Kesteren, and Dean Jackson. "The xmlhttprequest object." World Wide Web Consortium, Working Draft WD-XMLHttpRequest-20070618, 2007.

[Kester10] Anne Van Kesteren. "Cross-origin resource sharing." W3C Working Draft WD-cors-20100727, 2010.

[Khare04] Rohit Khare, and Richard N. Taylor. "Extending the representational state transfer (rest) architectural style for decentralized systems." In Proceedings. 26th International Conference on Software Engineering, pp. 428-437. IEEE, 2004.

[Koivisto03] Ari Koivisto. “Multi-User Publishing Environment Server.” M Sc. Thesis, Tampere University of Technology, p.54. 2003.

[Koskinen06] Kimmo Koskinen and Riku Suomela. "Rapid prototyping of context-aware games." In Proceedings of the 2nd International Conference on Intelligent Environments, p.8. Athens, Greece, 2006.

[Kurki10] Jussi Kurki and Eero Hyvönen. "Collaborative metadata editor integrated with ontology services and faceted portals." Workshop on Ontology Repositories and Editors for the Semantic Web, the Extended Semantic Web Conference, p. 5. 2010.

[Kuuskeri09] Janne Kuuskeri, and Tommi Mikkonen. "Partitioning web applications between the server and the client." In Proceedings of the symposium on Applied Computing, pp. 647-652. ACM, 2009.

[Lamport78] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system." Communications of the ACM 21, no. 7, pp. 558-565.1978.

[Lamport79] Leslie Lamport. "How to make a multiprocessor computer that correctly executes multiprocess programs." IEEE Transactions on Computers,100, no. 9, pp. 690-691. 1979.

[Lawrel12] Ryan Lawrel. “Steve Would Be Proud: How Apple Won The War Against Flash.” *http://techcrunch.com/2012/06/30/steve-jobs-war-against-flash/* (last accessed 27. March 2013)

[Lee04] Newton Lee. "Jabber for multiplayer flash games." Computers in Entertainment 2, no. 4, p.14. 2004.

[Liu73] C. L. Liu and James W. Layland. “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.” Journal of the ACM 20.1, pp. 46-61, 1973.

[Lowet09] Dietwig Lowet and Daniel Goergen. "Co-browsing dynamic web pages." In Proceedings of the 18th international conference on World Wide Web, pp. 941-950. ACM, 2009.

[Magnusson90] Boris Magnusson, Sten Minör, G. Hedin, M. Bengtsson, L. Dahlin, G. Fries, A. Gustavsson, D. Oscarsson, and M. Taube. “An Overview of the Mjølner Orm Environment”, In Proceedings of the 2nd International ConferenceTechnology of Object-Oriented Languages and Systems, Paris, 1990.

[Magnusson93] Boris Magnusson, Ulf Asklund, and Sten Minör. "Fine-grained revision control for collaborative software development." In ACM SIGSOFT Software Engineering Notes, vol. 18, no. 5, pp. 33-41. ACM, 1993.

[Marrin11] Chris Marrin. "Webgl specification." Khronos WebGL Working Group, 2011.

[McCarthy08] Phil McCarthy, and Dave Crane. ”Comet and Reverse Ajax: The Next-Generation Ajax 2.0.” Apress, p. 148. 2008.

[McFarland12] David Sawyer McFarland. “CSS3: The Missing Manual.” O'Reilly Media, p. 637. 2012.

[Meteor13] Meteor platform, *http://www.meteor.com* (last accessed 10. January 2013)

[Microsoft13] Microsoft Silverlight, *http://www.microsoft.com/silverlight/* (last accessed 2. September 2013)

[Mikkonen07] Tommi Mikkonen and Antero Taivalsaari. "Using JavaScript as a real programming language." Sun Microsystems Laboratories Technical Report TR-2007-168, p. 14. 2007.

[Mikkonen08] Tommi Mikkonen and Antero Taivalsaari. "Web Applications: Spaghetti code for the 21st century." In Proceedings of the 6th ACIS International Conference on Software Engineering Research, Management and Applications, IEEE Computer Society, pp. 319-328. Prague, Czech Republic, 2008.

[Minör93] Sten Minör and Boris Magnusson. "A model for semi-(a) synchronous collaborative editing." In Proceedings of the Third European Conference on Computer-Supported Cooperative Work, pp. 219-231. Springer Netherlands, 1993.

[Mogan10] Stephen Mogan and Weigang Wang. "The Impact of Web 2.0 Developments on Real-Time Groupware." In Second International Conference on Social Computing, pp. 534-539. IEEE, 2010.

[Mongo13] The MongoDB 2.4 Manual, *http://docs.mongodb.org/manual/* (last accessed 12. September 2013)

[MongoJS13] A Node.js module for mongodb. *http://github.com/gett/mongojs* (last accessed 2. September 2013)

[Mueller11] Rob Mueller. “HTTP keep-alive connection timeouts” *http://blog.fastmail.fm/2011/06/28/http-keep-alive-connection-timeouts/*, 2011 (last accessed 2. September 2013)

[MUPE08] “MUPE documentation” *http://web.archive.org/web/20081231130417/http://www.mupe.net/documents/* (last accessed 2. September 2013)

[Murugesan07] Murugesan, San. "Understanding Web 2.0." IT Professional 9, no. 4, pp. 34-41. 2007.

[NodeJS13] Node.js platform, *http://nodejs.org/* (last accessed 28. June 2013)

[Nokia03] Forum Nokia, “Overview of Multiplayer Mobile Game Design.” Forum Nokia, p.21. 2003.

[Nokia13] Nokia WidSets, *http://www.developer.nokia.com/Community/Wiki/Category:WidSets* (last accessed 25. February 2013)

[Nottingham05] Mark Nottingham and Robert Sayre. "RFC 4287-the atom syndication format." IETF Proposed standard, p.43. 2005.

[NowJS13] NowJS, *http://nowjs.com/* (last accessed 10. January 2013)

[Nunamaker90] Jay F. Nunamaker Jr and Minder Chen. "Systems development in information systems research." In System Sciences, Proceedings of the Twenty-Third Annual Hawaii International Conference on, vol. 3, pp.631-640. IEEE, 1990.

[O’Reilly07] Tim Oreilly. "What is Web 2.0: Design patterns and business models for the next generation of software." Communications & Strategies 1, p. 17. 2007.

[Oikarinen13] Jarkko Oikarinen. “IRC History”, *http://www.irc.org/history_docs/jarkko.html* (last accessed 2. September 2013)

[OpenCoWeb13] Open Cooperative Web Framework – Project intro. *http://opencoweb.org* (last accessed 2. September 2013)

[Oracle13a] Java For Mobile Devices, *http://www.oracle.com/technetwork/java/javame/javamobile/overview/getstarted/index.html* (last accessed 12. September 2013)

[Oracle13b] Synchronized Methods. *http://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html* (last accessed 2. September 2013)

[Paulson07] Linda Dailey Paulson. "Developers shift to dynamic programming languages." Computer 40, no. 2, pp. 12-15. 2007.

[Peacock00] Robert Peacock. "Distributed architecture technologies." IT Professional 2, no. 3, pp. 58-60. 2000.

[Perry07] Bruce W. Perry. “Google web toolkit for ajax.” O'Reilly Media, Inc., p. 40. 2007.

[Persevere13] Persevere, Documentation. *http://www.persvr.org/Documentation* (last accessed 20. March 2013)

[Popescu10] Andrei Popescu. "Geolocation api specification." World Wide Web Consortium, Proposed Recommendation, 2012.

[Prototype13] Prototype Core Team. “Prototype Javascript Framework.” *http://www.prototypejs.org/* (last accessed 2. September 2013)

[Queinnec04] Christian Queinnec. "Continuations and web servers." Higher-Order and Symbolic Computation 17, no. 4, pp. 277-295. 2004.

[Raggett99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. "HTML 4.01 Specification." W3C recommendation 24, p. 389. 1999.

[Ramakrishnan98] Satish Ramakrishnan and Vibha Dayal. "The pointcast network." In ACM SIGMOD Record, vol. 27, no. 2. p. 520. ACM, 1998.

[Resig06] John Resig. “Pro JavaScript Techniques.” Apress, p. 362. 2006.

[Richardson07] Leonard Richardson and Sam Ruby. “RESTful web services.” O'Reilly Media, p. 422. 2008.

[Richman87] Louis S. Richman. "Software catches the team spirit." Fortune 115, no. 12, pp. 125-136. 1987.

[Roseman96] Mark Roseman and Saul Greenberg. "Building real-time groupware with GroupKit, a groupware toolkit." ACM Transactions on Computer-Human Interaction 3, no. 1, pp. 66-106. 1996.

[Rossum94] Guido Van Rossum. "Python programming language." (1994) *http://www.python.org/* (last accessed 28. June 2013)

[Roth10] Gregor Roth, “HTML5 Server-Push Technologies, Part 1.” *http://today.java.net/article/2010/03/31/html5-server-push-technologies-part-1* (last accessed 2. September 2013)

[Ruderman01] Jesse Ruderman. "The same origin policy." (2001). *http://www-archive.mozilla.org/projects/security/components/same-origin.html* (last accessed 28. June 2013)

[Russel06] Alex Russell. "Comet: Low latency data for browsers." (2006) *http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/* (last accessed 28. June 2013)

[Saint-Andre09] Peter Saint-Andre, Kevin Smith, and Remko Tronçon. “XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies.” O'Reilly, p. 288. 2009.

[Schutzberg09] Adena Schutzberg. “How Google Maps uses the W3C Geolocation API and Google Location Services.” *http://apb.directionsmag.com/entry/how-google-maps-uses-the-w3c-geolocation-api-and-google-location-services/161053* (last accessed 2. September 2013)

[Shan06] Tony C. Shan and Winnie W. Hua. "Taxonomy of java web application frameworks." In Conference on e-Business Engineering, pp. 378-385. IEEE International, 2006.

[Shen02] Haifeng Shen and Chengzheng Sun. "Flexible notification for collaborative systems." In Proceedings of the conference on Computer Supported Cooperative Work, pp. 77-86. ACM, 2002.

[SocketIO13] Introducing socket IO. *http://socket.io/* (last accessed 30. June 2013)

[Stallman86] Stallman, Richard. “GNU Emacs manual.” Free Software Foundation, p. 354. 1986.

[Stirbu10] Vlad Stirbu. "A restful architecture for adaptive and multi-device application sharing." In Proceedings of the First International Workshop on RESTful Design, pp. 62-65. ACM, 2010.

[Sun06] Chengzheng Sun, Steven Xia, David Sun, David Chen, Haifeng Shen, and Wentong Cai. "Transparent adaptation of single-user applications for multi-user real-time collaboration." ACM Transactions on Computer-Human Interaction 13, no. 4, pp. 531-582. 2006.

[Sun08] “The Sun Labs Lively Kernel A Technical Overview”, *http://www.lively-kernel.org/development/media/LivelyKernel-TechnicalOverview.pdf* (last accessed 2. September 2013)

[Suomela04] Riku Suomela, Eero Räsänen, Ari Koivisto and Jouka Mattila. ”Open-Source Game Development with the Multi-User Publishing Environment (MUPE) Application Platform.” In Proceedings of the Third International Conference on Entertainment Computing, Lecture Notes in Computer Science 3166, pp. 308-320. Springer, 2004.

[Suomela05] Riku Suomela, Kimmo Koskinen, and Kari Heikkinen. "Rapid prototyping of location-based games with the multi-user publishing environment application platform." In Proceedings of the IEE International Workshop on Intelligent Environments, pp. 143-151. 2005.

[Taivalsaari08] Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls and Krzysztof Palacz. "Web browser as an application platform: the lively Kernel experience, Sun Microsystems." Inc., Mountain View, CA, p. 20. 2008.

[Taivalsaari11] Antero Taivalsaari, Tommi Mikkonen, Matti Anttonen and Arto Salminen. "The death of binary software: End user software moves to the web." In Ninth International Conference on Creating, Connecting and Collaborating through Computing, pp. 17-23. IEEE, 2011.

[Vaadin13] Vaadin 7 Wiki. “Broadcasting messages to other users”. *http://vaadin.com/wiki/-/wiki/Main/Broadcasting%20messages%20to%20other%20users* (last accessed 2. September 2013)

[Vaughan-Nichols11] Steven J. Vaughan-Nichols. “How Google–and everyone else–gets Wi-Fi location data.” *http://www.zdnet.com/blog/networking/how-google-8211and-everyone-else-8211gets-wi-fi-location-data/1664* (last accessed 2. September 2013)

[West12] Andrew G. West, Jian Chang, Krishna K. Venkatasubramanian, Insup Lee, “Trust in collaborative web applications”, Future Generation Computer Systems, Volume 28, Issue 8, pp. 1238-1251. 2012.

[White75] James E. White. "High-level framework for network-based resource sharings." RFC 707, p. 27. 1975.

[Wilde11] Erik Wilde and Cesare Pautasso. “Rest: from research to practice.” Springer Verlag, p. 528. 2011.

[William02] Stewart William. "MUD History", *http://www.livinginternet.com/d/di_major.htm* (last accessed 22. February 2013)

[Winer03] Dave Winer. "RSS 2.0 Specification." Berkman Center for Internet & Society at Harvard Law School, 2003.

[Winn09] Joss Winn. "PubSubHubbub: Realtime RSS and Atom Feeds." (2009). *http://seo-website-designer.com/PubSubHubbub* (last accessed 28. June 2013)

[Vogels09] Werner Vogels. "Eventually consistent." Communications of the ACM 52, no. 1, pp. 40-44. 2009.

[Wong02] William Wong. “Write Once, Debug Everywhere.” *http://electronicdesign.com/article/embedded-software/write-once-debug-everywhere 2255* (last accessed 6. June 2012)

[WoW13] World of WarCraft, *http://eu.battle.net/wow/en/* (last accessed 22. February 2013)

[Wood98] Lauren Wood, Arnaud Le Hors, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs et al. "Document object model (DOM) level 1 specification." W3C Recommendation, p. 212. 1998.

[You04] Yu You. “Situation Awareness on the world wide web.” Phd thesis, Jyväskylä, Finland, p. 171. 2004.

[Zhao02] Weiquan Zhao, David Kearney and Gianpaolo Gioiosa. "Architectures for web based applications." In 4th Australasian Workshop on Software and Systems Architectures, p 10. 2002.

[Zhao03] Weiquan Zhao and David Kearney. "Deriving architectures of web-based applications." In Web Technologies and Applications. Springer Berlin Heidelberg, pp. 301-312. 2003.

[Zukowski97] John Zukowski. “Java AWT reference. Vol. 3.” O'Reilly, p. 1045. 1997.

PUBLICATION I

Janne Lautamäki, Anssi Heiska, Tommi Mikkonen, and Riku Suomela. "Supporting mobile online multiuser service development." In the 3rd IET International Conference on Intelligent Environments. IE 07, pp. 198-204, 2007.

SUPPORTING MOBILE ONLINE MULTIUSER SERVICE DEVELOPMENT

J. Lautamäki[1], A. Heiska[2], T. Mikkonen[3], R. Suomela[2],

[1] TUT Institute of Software Systems,
P.O.Box 553 FI-33101 Tampere,
+358405198898,
janne.lautamaki@tut.fi
[2] TUT Institute of Software Systems,
P.O.Box 553 FI-33101 Tampere,
anssi.heiska@iki.fi
[3] TUT Institute of Software Systems,
P.O.Box 553 FI-33101 Tampere,
tommi.mikkonen@tut.fi
4 Nokia Research Center,
P.O.Box 100, FIN-33721 Tampere,
+358504835717,
riku.suomela@nokia.com

Keywords: MUPE, Eclipse plug-ins, multi-player Mobile games

Abstract

Mobile phones have emerged as the dominant communication platform. Since the early days, mobile phones have gradually become open platforms for applications and services, and more and more are expected of services. Multi-User Application Platform (MUPE) is a platform for rapid development of multi-user context-aware mobile application. This paper studies and analyzes which aspects of MUPE development are the most difficult and how to help developers with these problems. Moreover, an Eclipse plug-in toolset which solves at least some of these problems is presented. In addition example game development process with MUPE and toolset is introduced and analysed, and some feedback from MUPE workgroups is presented.

1 Introduction

Mobile phones have emerged as the dominant communication platform mainly due to the ease of one-to-one communications. Since the early days, mobile phones have gradually become open platforms for applications and services, and online services can already be accessed with mobile phones. However, creating a mobile online service is a complex task. Mobile networks have certain features that make them very different compared to landline networks. For instance, latencies in communication are higher, and roaming makes peak latencies even longer. These usually result in longer waiting times, but when accessing WWW, this is not fatal, just inconvenient. Perhaps therefore, apart from WWW services, we are yet to see an explosion of truly mobile services.

In comparison to conventional WWW services, mobile online multi-user services (e.g. messengers, online games) are even more challenging to implement, since several simultaneous slow connections can be available. Synchronizing data or user interface between different users is hard to implement in real-time. If one user modifies something in his view, it can take several seconds to distribute the modification to the other clients. This results in services being updated slowly. To overcome these issues, there are several ways to design services, as presented in for example. [7]

For a service to gain from the mobility dimension, it needs to be context-aware. Context awareness refers to applications ability to react to changes in the environment (e.g. [1]), and it introduces a further layer of difficulty into the service development. A good example is location; most mobile phones can be attached to a GPS device, or they even may include a built-in module. To use location in a service, there needs to be a set of APIs to access the information, and the information needs to be conveyed to other users of the service.

To simplify these problems, Multi-User Publishing Environment (MUPE) application platform [2] has been introduced for online multi-user services. The platform is accompanied with Eclipse-based tools for development, with an intention to ease the development of mobile online multi-user services.

In this paper we introduce the tools which are a set of Eclipse plug-ins whose goal is to lower the learning curve of beginners and helping experienced developers to complete MUPE services faster than before. The paper is structured as follows. Section 2 introduces MUPE at a level that is necessary for addressing the associated tools,

and discusses Eclipse and related plug-in development. Section 3 addresses the tools we have developed, and Section 4 provides a simple example on using the tools and some information on observed usability. Section 5 finally concludes the paper.

2 Background

2.1 MUPE platform

MUPE is an Open Source application platform for creating mobile multi-user context-aware applications. The platform can be used to create mobile games, virtual worlds, collaboration applications, and any other user authenticated services. MUPE is built entirely on top of Java (client-side on J2ME and server-side on J2SE).

The MUPE application platform has a client-server architecture, where the end-users use MUPE client to connect their mobile devices to the MUPE server in the internet. Any external information source in the Internet can be added to MUPE with the context components. The MUPE core connects the different parts of MUPE as illustrated in Figure 1.

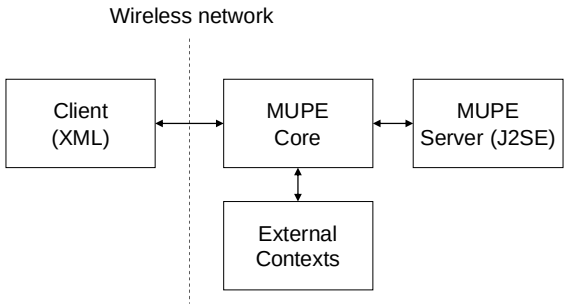

Figure 1. The MUPE application platform structure [5]

In comparison to traditional applications, MUPE applications are deployed differently. MUPE only requires a single client install, after which the user is free to browse new services, and use them with out any need for extra installations. In contrast, a typical Symbian or J2ME application needs to be first located, then downloaded and installed, and this is not always easy to do. When using python platform, the platform must first be installed on phone and then user has to do the same steps as with Symbian and with J2ME applications. Mobile phones offer help in finding and installing these applications, but still, the install process takes time.

In contrast to the traditional browser based approach, web browsers, MUPE platform has an access to the information supplied by the mobile phone, provided that it can be accessed from J2ME. For example MUPE clients can gain an access to Bluetooth, GPS, camera APIs, and other kind of services of the phone, something which is not available in web browsers. Furthermore, MUPE platform is also designed to enable easy creation of multi-user client-server applications, which are quite a demanding to create from the scratch.

Finally, MUPE applications also use less bandwidth when updating page than a regular web browser. A problem when using web browsers is that the entire web page has to be reloaded each time the user requests a change. AJAX partly solves this problem. With AJAX, every user action that normally would generate an HTTP request takes the form of a JavaScript call to the Ajax engine instead. If the engine needs something from the server in order to respond the engine makes those requests asynchronously. [2] This resembles a lot like ideology behind MUPE. MUPE still uses even less bandwidth, and there is also a possibility that the server takes active role and pushes data to client even without a request.

2.2 MUPE applications

As already shown in Figure 1, MUPE services are developed with J2SE for server-side functionality, and XML for client-side UIs and functionality. MUPE applications are written to be run by the MUPE server, which is the only component requiring programming in a basic application. The client user interface and functionality is created with client UI scripts that the client downloads from the server as illustrated in Figure 2. [2]

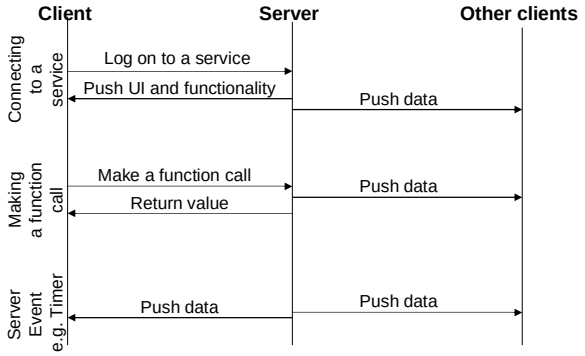

Figure 2. Client-server interactions

MUPE server-side J2SE development can be considered straightforward, because of the popularity of Java and availability of good tool support. However, MUPE client-side development is a different thing, because the XML that is written can be considered proprietary in its content, and mastering new language takes time. Furthermore, the lack of tool support has been a problem. While regular XML editors have been available, the problems remain that they cannot check connections between server and client -side, and they do not fully comprehend MUPE syntax because of some non-standard features. For example in MUPE platform there is a syntax to include XML files inside XML file by using `!#file.xml#!` notation. Instead, included XML files act just like their content would be placed where they were included.

Still, XML and related content is where MUPE excels. The XML allows the developer to create and modify the client functionality while the client is running, since the MUPE client interprets the XML at runtime. This allows very rapid service creation, where the developer can

constantly keep the client-server system running, while at the same time developing it.

XML files and Java-classes are both stored on the server-side. As soon as a user logs on to a server client acquires some of those XML file and interprets content of them to the user interfaces and functionalities to be presented for the user. After that the content of XML files is stored also on the client-side until it is deleted or sent again. Clients can make function calls to acquire XML content from server-side. For example `!#XMLfunction#!` written in an XML file calls *XMLfunction* function from server-side and returns its return value in to place were call originally was made. As a result, XML content stored on the client-side is changed. Server can use *PushData*–function to push data to active clients. Pushed data changes XML content stored on the client-side and also modifies user interface and functionalities. Data pushing can be result of server event (e.g. timer) or actions of other clients. It is also possible to read the content of XML file from server-side by using *getDynamicXML* function. Interactions are presented at figure 2.

2.3 Past experiences

In the past, we have organised several occasions for testing development of mobile services, as seen in for example [[2] and [2]]. These events have revealed several issues. In addition a foreseen issue is that true learning curve of the platform was been overly steep for some greenfield developer.

To be truly mobile services need to be context aware. They need to know the environment in which they are used, and the services need to react to this. Context-awareness, however, is not complex from application designer perspective. As an example, consider the following location can be expressed precisely in numbers, and an inaccurate location can be expressed as an area (circle, square or polygon). Obtaining this information is also easy in MUPE, since there are ready made scripts that can be used. The real problem is getting this far – since we are dealing with an application platform that has thousands of lines of code, it is imperative that the developers know the code at least to some extent.

There are several paths in learning sufficient skills with a certain platform, and development tools have proven to be a very successful way of reaching this goal.

2.4 Eclipse and plug-ins

Eclipse is an open-source platform-independent software framework. So far this framework has typically been used to develop Integrated Development Environments (IDEs), such as the Java IDE called Java Development Toolkit (JDT) and compiler (ECJ) that is delivered and deployed as part of Eclipse. MUPE uses J2SE for server-side and XML for client-side functionality. Eclipse already has a good IDE for Java development Toolkit and its design allows easy extensions by third parties. Because of that we chose to create a plug-in set with editor and a different kind of wizards to help developers with creating MUPE programs and editing MUPE XML format.

The design of Eclipse allows easy extensions by third parties. It provides possibility to make different kind of wizards; tools to manage workspaces; and to easily customize the programming experience. It was chosen because of JDT and good support for creating new editors and wizards. [6]

Eclipse has a wizard for creating new plug-ins so it is quite an easy task to create a new plug-in. Eclipse even has a set of template plug-ins. In MUPE plug-ins we have mainly used wizard and editor templates. Even if it is easy to create a new plug-in, it is not always easy to add right kind of functionality to it.

3 MUPE Tools

3.1 Background and motivation

As already discussed, the learning curve for making a new MUPE application has been steep, and the goal of MUPE toolset is to lower it as much as possible. Indeed, MUPE is easy and fast to use if you master it, but if you are beginner then it can be hard to accomplish anything in reasonable time just by reading tutorials by yourself.

Before starting the work with tools, several workshops had been organized in which people had tried MUPE and used it to make their own applications. Without MUPE toolset, it sometimes took the first three hours to get environment working on each laptop, and this leaves less time for service development. For effective MUPE development, the user needs Sun Java Wireless toolkit or other cell phone emulator for testing applications. User also needs Eclipse or other environment for developing Java-programs.

The previous workshops had revealed several key points to improve. There has been a lot of confusion about which java-methods or XML hooks could be called from which XML and how the XML files are related with each other. Also adding new pictures or new classes to the MUPE project had been consider being quite a hard. Not to mention to creating an entire new MUPE project. A goal of the toolset is to help people in doing the easiest and most straightforward tasks and thus leave more time for the harder ones. We have also hoped that the new toolset will courage people to learn MUPE by themselves outside any organized event.

A new MUPE project can be divided into three main areas:

1. Creating the service itself. A new service should be as easy to create as any Java project, since most tasks associated with this are mechanical (copying, etc.).

2. Creating the server-side. Each new content object inside the service should have support for easy instantiating.
3. Modifying the UI of the classes (XML) should support the special nature of MUPE projects.

For XML and Java there are already good tools available in Eclipse. The toolset needs to be built on top of these features.

3.2 Creating a new project

Before toolset available there were a lot of problems in creating a new MUPE projects. To get empty project to run in Eclipse environment user had to import all template files. Importing them is not as easy as it sounds because Eclipse import -wizard has a dozens of different kind methods to use depending in which format those template files are distributes.

Figure 3 presents the simplest possible MUPE application. Mupe application needs two external jars MupeCore and ContentClasses to work. MupeCore is for connecting MupeClient to MupeServer. And ContentClasses is server itself which is to be extended. At least four Java classes are needed to be implemented for server-side functionality, and for client-side functionality there also has to be some XML files.

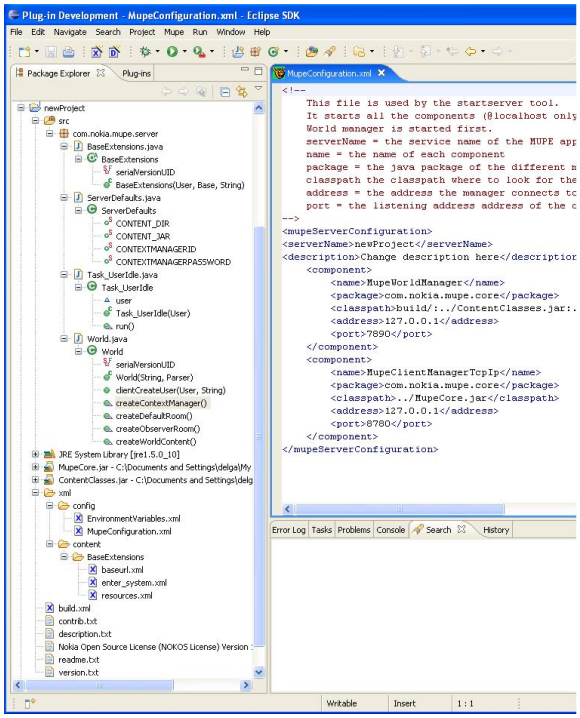

Figure 3. The simplest possible MUPE application (created with Minimum-project-template)

After several tries and failures user normally gets the template project in the package viewer of Eclipse but it is not all the user has to do. After importing the project the user has to link his new project to the MupeCore.jar and ContentClasses.jar files which he had to first download from the MUPE site. It is also possible that user has selected a wrong version from Java compiler. Of course when there is a workshop full of first timers there will be a lot of confusion.

With MUPE toolset user just selects file->new->"create a new MUPE project" from the menu. (figure 4) Then he just writes down the name of new project and selects which one of the several project templates he is going to use and then the new project is ready to roll.

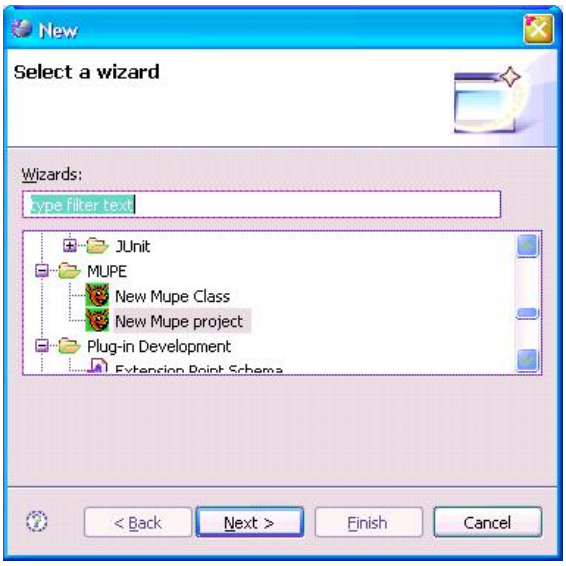

Figure 4. Select a wizard

3.3 Creating a new class

The basic MUPE server structure contains a World class for storing the content inside the application and four classes representing the content: User, item, Service and Room. The User class represents a single connected end-user in the system; the Room a location in the game world; Services provide add-on features for other objects and an item in any other data inside the application [2]. Normally all user created classes are extended from those four classes. For every class running on the server-side there has to also be a XML folder for the client-side functionality.

So creating a new Java-class for a MUPE project is not as straight forward, as with normal J2SE. For every class created user also has to create a folder and XML files needed. In figure 5 can be seen BaseExtensions class and BaseExtensions XML folder. Toolset offers easier way to create new classes with all things necessary. User just selects from which of the server classes a new MUPE class is extended. Tool can then offer the list of XML files there is available in the super class and it copies selected XML files from the super class to the new XML folder.

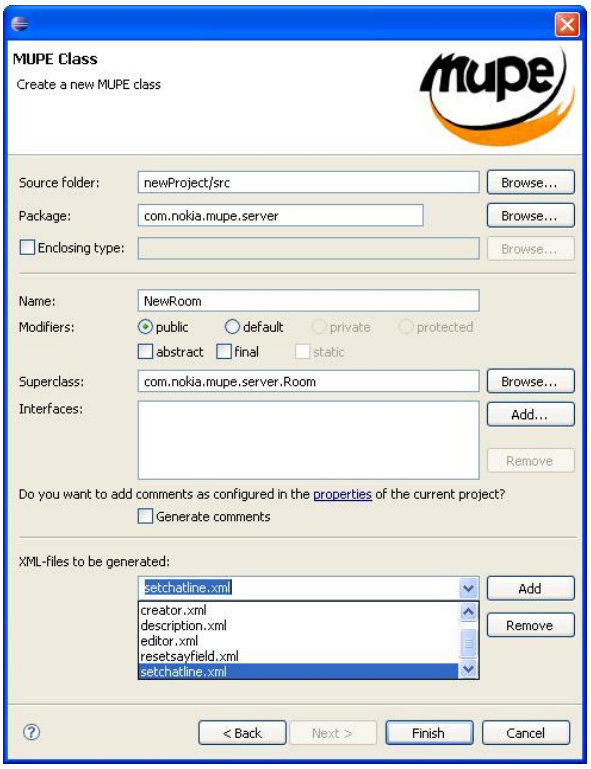

Figure 5. Create a new MUPE class

3.4 MUPE XML editor

Previously XML files needed for the MUPE programs were created and edited by using any regular text or XML editor available. Some of the editors were quite a good and some were not. With good editor like XMLBuddy it is easy to edit MUPE XML files. It finds out where tags start and where they end. It also helps user with attributes and other things available in normal XML. However, when using MUPE there are also things that regular XML editor can not find out and even some things that normal editor points out as mistakes or errors. For example: `!#file.xml#!` notation previously presented would be such.

There are also a different kind of notations for calling methods from Java-classes and sending information between client-side XML and server-side java files which regular XML editor can not check. MUPE editor knows which XML files are related to which Java classes. In figure 6 the user is trying to get content to text attribute. He has written: `!#` which have specific meaning in MUPE syntax. Editor notices what user is trying to do and offers completion proposals.

If user makes a little spelling mistake like in Figure 7 or something like it - it can be quite a hard to find it out, if he is using just regular editor. MUPE editor checks relations between files and reports errors found. It also checks that every tag has the beginning and the end. User can also use MUPE editor to jump to right file just with couple of clicks and see where methods or files are defined.

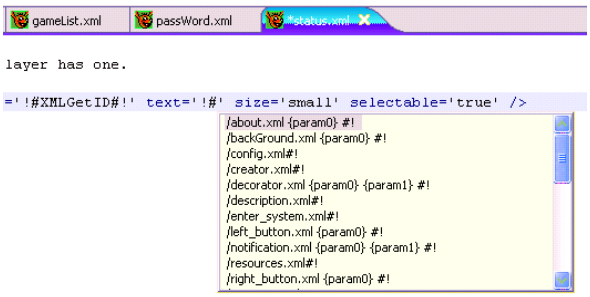

Figure 6. MUPE completion proposals

Toolset also helps user to refactor his project. One of the common mistakes when using MUPE is that when user changes the name of the class, he forgets to change name of the XML folder. Toolset also has some extensions for JDT. It can for example check if file, which *getDynamicXML* function tries to call really exist.

Figure 7. Editor has found a problem from a resource.

We have also created a set of code snippets to use. Before these snippets developer had to check from MUPE website how the certain feature has to be implemented. Off course most common features could also be remembered, but it is quite a boring to write same text dozens of times. These snippets try to cover all the most commonly used functionalities so the user can just select what kind of functionality he wants to implement and then snippet is copied to the right place.

3.5 Experiences

There are a lot of articles about Eclipse concerning its architecture and low level interfaces. Something between them is missing. However, to implement certain functionality the developer has either to extend an available extension point or to extend an Eclipse class. For this process we could not find enough material. There were a lot of examples on how to create a simple plug-in, but not many examples on how to create more complex environment were found.

Due to the above, it usually takes a lot of time to find which class to extend or which extension point to use. Sometimes naming conventions of Eclipse are not so easily followed or understood. For example there are classes called *IWorkbenchWindowPulldownDelegate* and *IWorkbenchWindowPulldownDelegate2*. There is a certain difference between them, and the developer has to know which one of them to implement.

4. Usability tests

4.1 Case Example

> In this project we have created wide variety of little games and applications and a couple of larger ones. One reason for creating these applications is to test usability of our tools. One example game is Dung Beetle. The idea of Dung Beetle is that every player has a dung beetle[1] to control. Many dung beetles, known as rollers, are noted for rolling dung into spherical balls, which are used as a food source or brooding chambers.

It took about 8 hours to create this two player game. In first four hour a new project was created using the BattleBoard-project-template. Some pictures were drawn with Gimp, and user interfaces were designed and implemented. Thus, after the first four hours of implementation, we had a single playable prototype of the game. Beetle could be moved, and dung could be rolled. In the next four hours the game took its final form. The game was modified to a multi-playable form. Timers to drop more dung on the board were added, and the winning condition was checked. First test game was also played. Figure 8 presents the final user interface of the game.

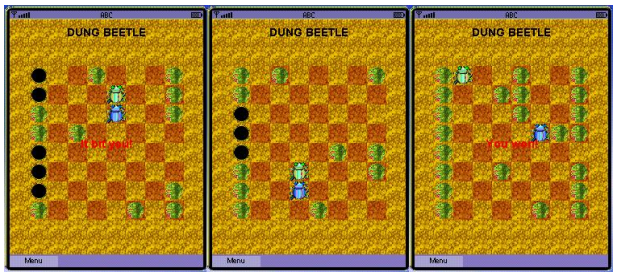

Figure 8. Three different game situations.

The total number code lines at the server-side was 314. 105 of those come from battleBoard-template, which contains two rooms. One is a lobby room where players enter when logging on, and another is a game room, where the player can move from lobby room and which contains a tile map and selector. Extending battleBoard-template to Dung Beetle took about 200 lines of server-side Java code.

341 lines of XML were needed for client-side UIs and functionalities. 255 of those come from template, so the developer had to write about 90 lines of XML. Those 90 extra XML lines contain moving the beetle, pushing dung piles, biting other beetle, checking winning conditions, high score listing and collision checks. 200 lines of added Java contain timers for dropping more dung and for biting, high score listings and double player capabilities and interactions between players.

[1] Dung beetle refers to those beetles which feed partly or exclusively on feces.

4.2 Feedback

We have arranged several MUPE workshops after starting of the tool development. In most of them, we have collected some feedback about toolset. Still the number of answers has been quite a limited, and the questions we have asked have varied from one inquiry to the next so there is no possibility to make any statistical analysis from them. Nevertheless, all participants who had given feedback have said that toolset was useful. Most of the people who had new ideas for tools had hoped for graphical user interface editor.

Still, it is pretty certain that beginners at least thought that toolset helped them. We have also monitored users in those workshops and we have noticed that toolset not only helps them but it also gives ideas what kind of applications can be created and what kind of functionalities can be made. Project templates give them a hint what kind of projects it is possible to create and code snippets give ideas about possible functionalities.

Based on our own and colleagues experience, it is quite a certain that the toolset also speeds up and helps the work of the experienced MUPE developers. Common mistakes and failures in logic are eliminated because of the editor. Commonly used functionalities are available as code snippets and code completion makes it faster to write text.

5. Conclusions and future development

MUPE platform enables rapid creation of multi-player mobile games. But because of the client-server architecture and proprietary XML format, the learning curve for making new MUPE applications is quite a steep. The goal of the toolset was to lower learning curve as much as possible. Toolset should also help experienced developers to complete MUPE services faster than before.

This paper presented some of the problems facing those who tried to develop MUPE applications before toolset existed. It also presented a set of Eclipse plug-ins that can support MUPE application development. Toolset should have solved at least some of the problems in MUPE development previously presented.

When creating MUPE applications and organizing workshops we have noticed that some tasks in creating new MUPE applications are more difficult than others. MUPE applications have a certain structure and we have noticed that for users who use MUPE at first time it is quite a difficult to start a new MUPE project. Because of that we have automated this part of application development. The software developers of today do not like to use just plain text editor when creating complex software systems. They demand an editor that has features specifically designed to simplify and speed up input of source code.

As for the future, we have thought that a WYSIWYG editor for creating and editing client-side interfaces would

be quite a nice. Eclipse has a good support for making graphical editors. Eclipse has a framework called GEF (Graphical editing framework) which is intended be used to create that kind editor. Problem is that making WYSIWYG editor is quite a demanding task and there is a possibility that in short-term, there are no resources for making it.

Another problem with WYSIWYG editor would be that in MUPE XML syntax elements are normally anchored to each other. When we would drag and drop an item to the view, we should tell editor how a new item is related to the old ones. Editor should know to which of the existing items we are anchoring the new item. It should also know to which part of the new item is anchored to which part of the old one. For example we can anchor a centre of the new item to a bottom left corner of the existing one. Thus, adding a new XML element would not be straight forward WYSIWYG editing. One solution for this editing problem could be to create a tool, which would have a preview window but not editing capabilities, but anyway, many of these XML elements are created at runtime so view would not be complete.

Acknowledgements

We would like to thank the entire personnel of the MoMUPE project, and TEKES (Finnish Funding Agency for Technology and Innovation) for support.

References

[1] A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, 2000.

[2] J.J. Garrett. *Ajax: A New Approach to Web Applications* http://www.adaptivepath.com/publications/essays/archives/000385.php

[3] K. Koskinen, R. Suomela. *Rapid Prototyping of Context-Aware Games*, In proceedings of the 2nd International Conference on Intelligent Environments (IE06), July 5-6, Athens, Greece, 2006

[4] R. Suomela., K. Koskinen, K. Heikkinen. *Rapid Prototyping of Location- Based Games with the Multi-User publishing Environment Application Platform.* Proceedings of The IEE International Workshop on Intelligent Environments, June 2005, 143-151.

[5] R. Suomela, E. Räsänen, A. Koivisto, J. Mattila: *Open-Source Game Development with the Multi-User Publishing Environment (MUPE) Application Platform.* Proceedings of the Third International Conference on Entertainment Computing 2004, 308-320, Lecture Notes in Computer Science 3166 Springer 2004

[6] *Eclipse (software)* http://en.wikipedia.org/wiki/Eclipse_%28software%29

[7] *Overview of Multiplayer Mobile Game Design*. Available online at: http://www.forum.nokia.com/main.html (2004)

PUBLICATION II

Anssi Jääskeläinen and Janne Lautamäki. "Analyzing context-aware service development under MUPE platform." In the Eighth International Workshop on Applications and Services in Wireless Networks. ASWN'08, pp. 26-34. IEEE, 2008.

Analyzing Context-Aware Service Development under MUPE Platform

Anssi Jääskeläinen
Lappeenranta University of Technology
jaaskela@lut.fi

Janne Lautamäki
Tampere University of Technology
Janne.lautamaki@tut.fi

Abstract

Context-awareness indicates that a computer system is linked with changes in the real world. Context information can be used for numerous different purposes, but the real break through is still pending. Context information is useful only when it is not static, and hence in desktop computers there is very limited need for context awareness. In mobile phones situation is different. MUPE is an Open Source application platform for creating mobile multi-user context-aware services. It is based on client-server architecture and both server and client are capable context providers. A typical client context would be GPS location or Bluetooth ID. Typical server context would be information like weather or stock market. This paper introduces context-aware services made by professional and non-professional and compares those against each other to point out that context-awareness under MUPE is easily and rapidly utilizable even for non-professionals developers.

1. Introduction

Mobile phones have emerged as the dominant communication platform. Since the early days, mobile phones have gradually become open platforms for applications and services, and online services can already be accessed with mobile phones. Mobile networks have features that make them very different compared to traditional networks. In particular, latencies in communication are higher. This usually results in longer waiting times, but when accessing World Wide Web (WWW), this is not fatal, just inconvenient. In comparison to WWW services, mobile online multi-user services (e.g. messengers, online games) are more challenging to implement. Synchronizing data or user interface between different users is hard to implement in real-time since several different slow connections may exist. If user sends data to the server, it can take several seconds to push the information to the other clients. This results in services being updated slowly. Perhaps therefore, apart from WWW services, we are yet to witness an explosion of truly mobile services.

One way to gain from mobility dimension is to create services that are context-aware. Context-awareness refers to services' ability to react to changes in the environment [1]. This although introduces a further layer of difficulty into the service development, since context-awareness requires more functionality from both the client and the server side. A good example of context-awareness is location; most mobile phones can be attached to a GPS device, or they even may include a built-in module. To use location in a service, there needs to be a set of APIs to access the information, and the information needs to be conveyed to other users of the service.

Context awareness is not yet very well known concept. Most people think that mobile applications are like desktop applications, but more difficult to use and install. GPS navigators are the only commonly used context aware applications. However there is no real reason why there could not be wide variety of applications using and sharing contexts.

Another important factor is persistence, which means that the service is running all the time and the user has constant access to it. For example in persistent multi-user games such as Travian [2] it is important to be aware if someone is attacking.

In this paper we present some persistent context-aware services made by both professionals and non-professionals. When we are speaking about professionals and non-professionals in this paper we are only meaning MUPE, which means that non-professional MUPE developer might be a professional Java developer.

Main point of this paper is to show that it is possible to use multiple different contexts for many variable purposes under MUPE. We are also trying to

make clear that it is so easy that even a non-professional developer can benefit from the power of context awareness. Paper also contains a comparison between the main differences/similarities of approaches and implementations to demonstrate the introduced point.

The rest of this paper is structured as follows. Section II introduces MUPE and tells about the context-awareness. Section III presents one service made by professional and one service made by non-professional, then section IV compares these developed services against each other and shows few reference services to point out the power of context-awareness. Section V finally concludes this paper.

2. Context-awareness under MUPE platform

To overcome the introduced issues of mobility and context-awareness numerous ways to design services have been proposed [3]. In this paper we use an application platform called MUPE (Multi-User Publishing Environment). MUPE is an open source platform for creating mobile multi-user context-aware services. It is mostly aimed for relatively small projects. In huge projects with good resources and professional programmers there is no need for platform at all. In projects with bigger developer teams it is a lot more convenient to just use Symbian C++ or J2ME and create application from scratches, because with platform there always comes some restrictions.

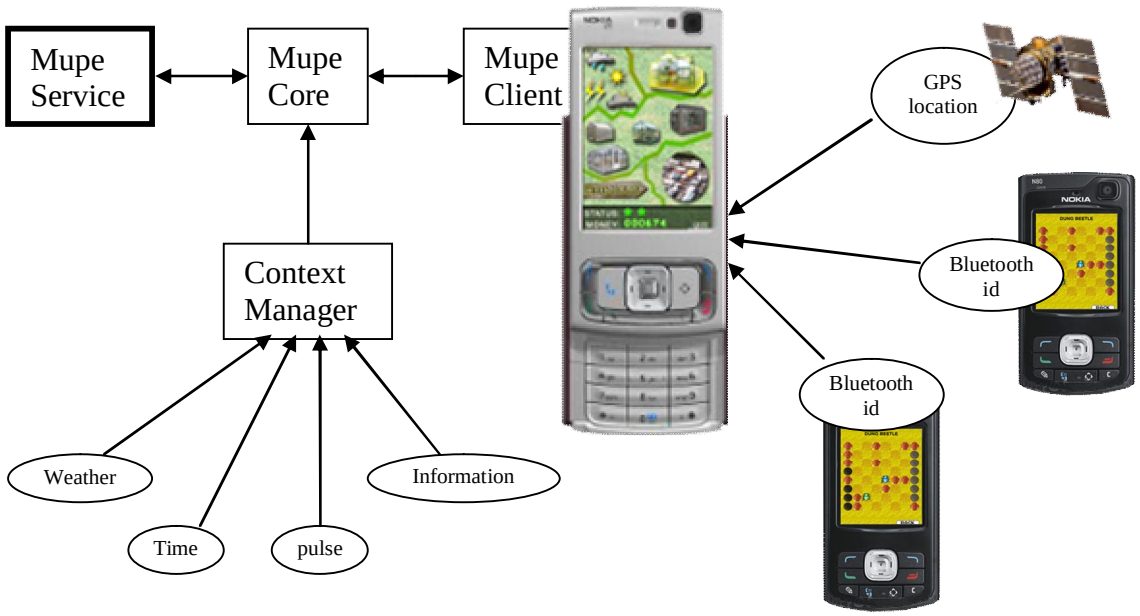

Fig. 1. The MUPE application platform structure and ways to get content

2.1. MUPE

MUPE Platform is based on client-server architecture and it can be used to create e.g. mobile games, virtual worlds, collaboration applications, and any other user authenticated services. MUPE is built entirely on top of Java (client-side on J2ME and server-side on J2SE) [4]-[6]. End-users use MUPE client, which is just one installation to the phone, to connect their mobile devices to MUPE service. This service is actually a package that also contains MUPE Core components and context manager but commonly this whole package is called just as the server or service. Figure 1 illustrates this structure in the upper left corner. In the following we introduce the main components.

- MUPE Service is the actual service, game, etc.
- MUPE Core connects clients to the service and makes contexts offered by Context Manager available for the service. Usually developers do not have to change anything inside the Core and new features to it can be requested from the Core developers.
- Context Manager brings external information to the service through the Core. It is specialized to handle external information sources such as the Internet or XML feed.
- MUPE client is applications installed on mobile phone. It is used to browse MUPE services over wireless network.

Service, Core and Context Manager normally runs on a single web server, but since they are separate components, they can be distributed to different servers if need arises.

With its modular structure MUPE combines the best parts of stand alone and browser based applications. In comparison to stand alone applications, MUPE services are deployed differently. MUPE only requires a single client install, after which the user is free to browse new services, and use them without extra installations. In contrast to the browser based approach, MUPE client has also an access to the context information supplied by the mobile device, and so the MUPE services are not tied only to context information available in internet.

2.2. Context-awareness

Context-awareness indicates that a computer system is somehow linked with changes in the real world. [7] Context information can be used for several different purposes. Under MUPE platform contexts can be divided into a two different categories with

TABLE I
Client side context types

Context	Description	Class	Service
Camera	Many services use camera to add personality to user interfaces, identify different items or just capturing images that can be shared with other users.	Device context	Detector, CamQ [9]
Bar code	A form of using camera on such devices that supports barcode reading. If barcode reading is not supported by the device it is entered by hand.	Environmental context	Product control [9]
Bluetooth ID	Bluetooth can be used to detect the devices that are nearby. Bluetooth is available in most phones, so it can be used freely in all designs.	Device context	Insectopia, Sandman [9]
GPS	Any device with Bluetooth can use an external GPS device. There are also few devices on the market that has build in GPS device.	Spatio-temporal context	Traveller [9]

TABLE II
Server side context types

Context	Description	Class	Service
Search engine	Uses search engine hit count to do something in the service.	Service context	Netbubble [9]
Stock market	Retrieves information about the stocks	Service context	Stock system [10]
Weather	Weather information can be used for example to show weather, modify climate or maybe change the appearance of user interface.	Environmental context	Greenhouse [9]

different roles (Fig. 1); device (client) context (Tab. I) and server context (Tab. II). Client side context-awareness requires from the device, that it has ability to collect information about its environment. So client can monitor the real world around the user and accesses information stored inside the phone e.g. GPS location or Bluetooth ID. Role of the server is to provide contexts from Internet e.g. weather, stock market, etc.

It is fairly easy to get one context and use it as an element in a game or service. One can create games that are based on collecting Bluetooth ids or mapping locations with GPS. It is also possible to combine the server side context the client side context. For example a client can determine its location using GPS, a server can download a weather report, and the two contexts can be combined to the temperature in the current location of the user. Another example could be speed dating. Server side has some stored information about preferences of the users and client side uses bluetooth ID to locate other clients nearby. When two clients are nearby, the server compares preferences of users, and if they match then associated users can start speed dating.

It is possible to use wide variety of different contexts inside MUPE services. Contexts available at client side are limited to those accessible from J2ME. Though, under series 60 phones it is possible to implement a context provider plug-in that can access contexts that are normally only available for Symbian applications. At the server side only limit what can be used as a context is imagination.

2.3. Classification of context

Context information can be used for several different purposes according to their classifications. One classification is presented by ePerSpace [8] which categorizes different contexts into the following groups: environmental context (e.g. temperature), personal context (e.g. heart beat), task context (e.g. events), social context (e.g. social network), spatio-temporal context (e.g. location, time), device context (e.g. Bluetooth id), service context (e.g. service specific data) and Access context (e.g. network capabilities). Table 1 and Table 2 present some of the contexts used under MUPE and classifies those according to ePerSpace categorization.

2.4. Persistent applications

Persistence means that the state of the application continues its evolving and time passes even when users are not online. For example in “World of the World Craft” which is currently the most popular massive multiplayer online role playing game, this is done by using client-server architecture. So game world is running at the server-side and users use special client to log in and log out when ever they want.

It is also possible, but not common, to create persistent single player game worlds without server-side. One example of that is Nintendo GameCubes Animal Crossing [10], which slogan is: "It's playing, even when you're not". Idea is that when user quits the game, a time stamp is stored on memory card. When user continues playing the game checks from GameCubes clock how much actual time has passed and generates calendar evens about what has been happened while user was away.

2.5. Creating context-aware persistent services

With MUPE, professionals and non-professionals can easily create context-aware services. Eclipse based tool [11] [12] offers snippets and other ways to make implementing of context-aware services easier. Client side context-awareness is easiest thing to implement, developer just grasps code snippet and then places it in one of the XML files, after that the context information is ready for use. Server side contexts are little bit harder; snippet has to be modified to get it work, since user has to select web site which is used as a context provider and then he has to implement methods to parse right data from the site. Persistent aspect comes from the fact that the server is running all the time and users can connect and disconnect as they want. It is just up to the developer if persistency is exploited in the service. MUPE offers both context-awareness and persistency almost by default, so problems are not in using those, but in inventing a fun and intuitive usage.

3. Selected services

To demonstrate the power of context-awareness we have selected two different services for thorough testing. In this comparison both services are camera games, which have been chosen since camera already have wide penetration in mobile environment and camera works reliable under MUPE. Games are selected, since most of the MUPE services developed are games, but equally test services could be general applications. Both of these games are relatively small in size, and they are created by quite a small team.

Fig. 2. Screenshot of CamQ.

Fig. 3 Screenshot of Wizard Card Game.

3.1. CamQ

In CamQ player uses camera to create a picture quizzes for the other players. Quizzes are created by taking pictures and describing them using three words. It is also possible to give a hint or a comment for the picture. Player should try not to make quiz too obvious or too hard, because the goal is to get equal amount of correct and incorrect answers from the other players, who are doing everything they can to come up with the three correct words related to picture! (Fig. 2)

This game was developed in The Helsinki Institute for Information Technology (HIIT) and its visual outlook was created by graphical designer. Game went thru several different stages including focus group sessions and multiple functional prototypes. First version was based on old finish television quiz "Kymppitonni" where players tried to guess a word based on verbal hints given by another player. It was a good and working demo, but the problem was that it was synchronous, so there had to be four players present at the same time. This caused problem since sometimes in real life, it is difficult to find players online. Little by little CamQ was pushed towards asynchronous game mode with no waiting and no real time actions between the players.

Final published version of CamQ does not limit the number of participants in any way and it is totally asynchronous. So it is online multiplayer game but there is no constant need for other online players. Other players are just needed to create more content to game and to solve quizzes other players have created.

3.2. Wizard Card Game

In Wizard Card Game player uses camera to collect up to 20 pictures and uses those as an army. Game logic determinates attack and defence values for monsters by calculating the hash value of the pictures. After that the army is created and player can use it to attack against other players (Fig. 3).

This game was developed as an exercise work on game development course which took place at the Tampere University of Technology. In the beginning of the course there was two hours hand on demo about using MUPE platform. Game theme and mechanics was also discussed with course staff, but afterwards students did not need any other help to finish their exercise work, except MUPE WWW site.

Game itself is essentially two player game, but any number of players can join to create an army and there is no theoretical limit about how many simultaneous duels can be going on at the same time.

4. COMPARISON OF SERVICES

In this comparison we demonstrate that there are no significant differences between the context-aware services developed by professionals and non-professionals when speaking about operability. Of course we do not claim that MUPE changes non-professionals to professional but this similarity is our key point in proofing that it is easy even for non-professional developers to create context-aware services in a relatively small time with MUPE. On the other hand this comparison also states the obvious fact that there are big differences between the services although they are not necessarily visible to end users.

In generally, comparison of MUPE services is rather difficult thus they are usually based on a different logic, graphics etc. but it is not an idea of this comparison to study general service elements. We are just pointing out that even a MUPE first timer can create a context-aware service in just few days despite the fact that it may sound difficult.

4.1. Used metrics and tools

For understanding the differences between the services in a code level we have collected several different metrics. We used simple LOC (Lines of Code) measurement separately for XML and Java and also calculated classes, methods and attributes inside the service. Metrics were collected with Eclipse plug-in called Metrics [13]. These values shows from high level of view what the services look like. After these simple values we used McCabe's Cyclomatic Complexity calculations to go beneath the surface of the service. As a summary of McCabe it can be said that any complexity above ten is likely to cause problems in testing and maintaining and that way will require fixing [14].

We have reverse engineered UML charts for games with eUML2 plug-in [15] inside eclipse. The reverse engineering is always a bit inaccurate and therefore the produced charts have been slightly modified. Also all unmodified classes have been removed from the images just to keep those simple.

4.2. Results

In CamQ camera is really important game element and without pictures game could not exist. Player needs to carefully select picture to get right answers from other players because they make their guesses totally based on the picture. In the Wizard Card Game camera is just a kind of random element. It is nice to have possibility to custom your cards with own pictures, but still it is just some kind of spice. One could argue that strength of card is based on picture, but in this case it is not good argument thus Wizard Card Game uses a simple hash function to transform image to numbers and it is quite random. Of course it is possible to develop better algorithms that could e.g. calculate colour or brightness.

Simplified UML charts of these services only

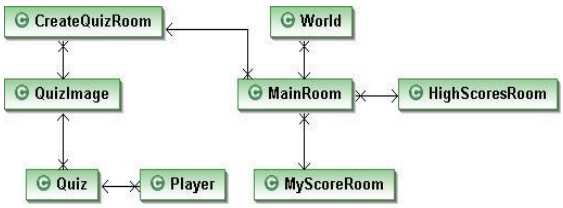

Fig. 4. UML of CamQ

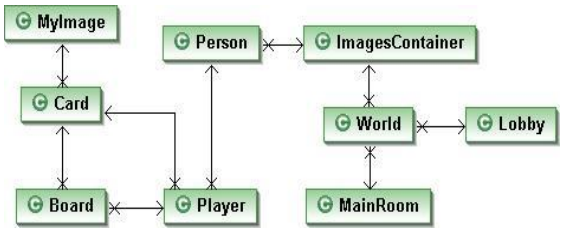

Fig. 5. UML of Wizard Card Game

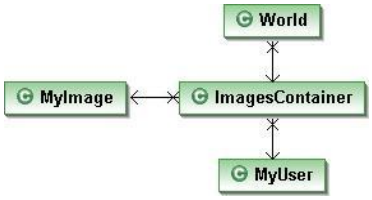

Fig. 6. UML of Camera Template

TABLE III
Complexity Comparison for camera services

	Camera Template	CamQ	Wizard Card Game
Number of Attributes	3	22	28
Number of Classes	5	10	10
Number of methods	18	165	98
McCabe Cyclomatic compexity average	1,111	2,107	2,173
McCabe Cyclomatic compexity max	2	7	33
Total Lines of Java Code	104	1650	1020
Total Lines of XML code	188	5280	557

consider the actual service and it does not take into account the interactivity with the client. Point of these charts is to prove that even the main structure of developed services looks approximately equally complicated. (Fig. 4&5)

Similarities between the CamQ and Wizard Card Game are mostly due to MUPE, since it automatically creates a certain skeleton for the services. Camera Template (Fig. 6) has been used as a base for both of these complete games. All classes in Camera Template have counterparts in the example games. E.g. MyUser of Camera Template is named as Player in both of them. MyImage is named as QuizImage in CamQ. Camera Template has just one room: "ImagesContainer", while both of the examples have several rooms. Developers can of course modify or even replace pre generated skeleton but usually there is no need to do that.

Although the high level structure is similar, services created by professionals are usually more complex in functionality than the ones made by non-professional. In Table III we present some metrics collected from both our test services and from Camera Template.

When carefully comparing Table III and UML charts (Fig. 4-6) it can be noticed that in Table III there are 10 classes in both example projects but in the charts there are only 8 and 9 classes. This is due the few simplifications we have made for the UML charts. For example because of special nature of BaseExtensions class we left it away from all the UML charts.

The couple of first metrics look quite a like for both of the examples. This is mostly due the template used. Greatest difference in the measurements between the services is the total number of XML code lines, which CamQ has about ten times more than Wizard has. Some of these lines come from the eye candy which is the visible form of difference between professional and non-professional but there is also the invisible part that mostly consists of XML scripting language. When using MUPE it is possible to create functionalities for client-side using XML scripting. XML scripting is a powerful tool and it makes services run a lot faster, because they do not have to make so many slow server calls over hi-latency wireless network. However MUPE XML is proprietary format and there is quite a steep learning curve for learning its full potential.

When comparing number of the XML lines to number of the Java lines we notice that in CamQ there are about three lines of XML for every line of Java code. In Wizard Card Game the relation is opposite. There are two Java lines for every XML line. Professional MUPE service developer is reducing the number of server calls by using XML scripting while amateur designer is tying to minimize the need of XML because he already knows how to do things with Java.

XML file written by professional developer typically contains simple functions such as checking the device resolution and doing some calculations to reduce the server load and this is also the case in CamQ vs. Wizard Card Game. CamQ has also about twice more Java methods than Wizard Card Game has.

Another noticeable difference between these services is the maximum complexity value, which is really high in a couple of classes inside the Wizard Card Game. These high values can be tracked into those functions which controls AI (complexity 33) and attack (complexity 24). High values are probably due to lack of planning, thus developers had to create working functionality in a short amount of time. Much of this complexity is due to multiple exceptions which are handled by using IF –statements although there are better and less complex ways to do this.

When we checked the services out without UML or metrics we noticed that there are lots of differences that are mostly invisible or hard to notice to the end users. Context dependable difference is the way how retrieved contexts are used inside the service. Professionals prefer to build their own context-aware functionality instead of using those offered by MUPE and they also try to use retrieved context inventively instead of repeating old ways. This can be noticed in richer user interfaces and versatile functionalities. E.g. in a camera case, output is usually something more than just image on the screen. Non-professionals tend not to use contexts at all if they are not somehow forced to do so. This can be typically explained by the lack of knowledge and by tendency to create conventional desktop applications for mobile environment.

Professionals try to develop services that work in every situation. For instance in our test case both services look fine when using the resolution they are designed for. In the Figure 3 it can be noticed that Wizard Card Game does not occupy the whole screen. This is because Wizard has not been designed for 240x320 resolution and it does not contain functions for scaling graphics. CamQ on the other hand is designed to work with all common screen resolutions

As a reverse side, it can be said that services made by non-professionals tend to work better as they are

less complex and offer less functionality that may cause malfunctions or problems. Generalization of course is impossible because there were really good and working services in those made by professionals and also really bad non working services under those made by the non-professionals.

4.3. Reference services

To widen the results of our study and to get a reference point for thoroughly tested services we have included few additional services. Services in Table IV are developed by non-professionals during a code camp sessions and services in Table V are made by professional developers.

Code camps are one form of group working, which lasts about one week. During a code camp a group has to design and implement an application with given properties. It has been a common habit in the Lappeenranta University of Technology code camps that groups help each others although there has always been a competition between the groups about the best idea & application. Group also gets help from the instructors.

Noticeable in the services developed under the code camps, is the fact that these have been designed, implemented and tested in a very short period of time. In our cases groups have four days to design in implement a working service and the actual implementation process took place during the last 48 hours. For this reason it is justified to assume that the appearance, used techniques and quality are no where near of those developed but the professionals, but when services were returned we noticed that the assumption was mainly wrong. This proves that it is easy to create fully functioning services with MUPE in a rapid manner, but if one wants to be absolutely sure that game works in all situations and with many different phone models it takes a lot more time and effort.

Tables IV and V shows the values for selected services and by comparing these two tables with table III we can notice few similarities and differences. Main similarity between the two groups of services is that the average complexity among professional developers is lower than in those made by non-professional and same also goes to maximum complexity value. Of course there are exceptions such as TankedSW which has the lowest average complexity after the Camera Template, which can be explained with the fact that complexity is only dependable of Java and does not include dependences to the client device. So if you know Java well enough are already half MUPE professional. Another difference can be noticed in the LOC values which in generally is higher in the services developed by professional. Exception is Dung Beetle which is a really small game developed by a highly qualified MUPE professional.

Apart from these few noticeable differences there is

TABLE IV
Complexity comparison non-professionals

	MSS	Tanked SW	Face book	Tourism Essen-tials
Number of Attributes	5	32	1	8
Number of Classes	7	10	6	8
Number of methods	24	71	39	27
McCabe Cyclomatic compexity average	2,25	1,324	1,821	1,815
McCabe Cyclomatic compexity max	17	14	7	7
Total Lines of Java Code	296	446	512	308
Total Lines of XML code	205	680	412	334

TABLE V
Complexity comparison professionals

	Product Control	Net Bubble	Sand man	Dung Beetle
Number of Attributes	176	104	36	15
Number of Classes	59	32	9	9
Number of methods	441	192	92	43
McCabe Cyclomatic compexity average	1,482	1,4	1,978	1,465
McCabe Cyclomatic compexity max	8	7	11	6
Total Lines of Java Code	4798	1833	997	326
Total Lines of XML code	2692	1410	1067	391

not much else that can be noticed with the numerical comparison. Invisible differences are mostly the same as in CamQ vs. Wizard case. Every service in table V developed by professionals support at least two different resolutions and e.g. NetBubble four different resolutions. Another difference is again the functionality of XML. Services developed by non-professionals only contain the needed parts and nothing to reduce the server load. Since we did not find any major differences in the tested services, we came in to a conclusion that MUPE platform offers easy and rapid way even for non-professional developers to implement context-aware mobile services.

5. Conclusion

Multiple different services with context-awareness have been built by multiple different authors using MUPE platform with or without Eclipse-based tools. Nothing restricts development process to Eclipse but if certainly makes the development process easier than using other tools without MUPE support. Existence of the MUPE tools also makes it really simple to build a new MUPE services from the scratch, thus tools generate the basic environment skeleton automatically and developers only need to add their own functionalities to the code.

This can also be noticed when comparing the services made by professionals to those made by the non-professionals. Of course this superficial study and short descriptions of services would not tell the whole truth, but the quality of services made by non-professionals is generally about the same than those made by professionals. This strangeness can be explained by the fact that non-professionals only add few simple own functionalities to pre generated skeleton and are less likely to break something. It is also possible that non-professional under MUPE is a professional in Java which all ready makes him half way to MUPE professional.

As a result of our study we confirmed that it is possible for non-professional service developers to design and implement quite impressive context-aware services in a short amount if there are professional instructors available. If instructors are not available it is still possible for non-professionals to create context-aware services easily but the quality is more dependable of developers' knowledge of Java and XML. Differences between contexts used by professionals and non-professionals are minor but it is still obvious that functionalities offered by MUPE tools are more commonly used by non-professionals thus they offer easy way to add context-awareness to service. Professionals prefer to build their own context-aware functionality or modify those offered by MUPE tools and use the available context more inventively.

MUPE tool that was mentioned earlier formed one problem to this comparison since it contains a template projects for camera and Bluetooth. The effect of these templates is that students rather use camera and Bluetooth as a context in their projects than try to use other available contexts with no ready made template.

Acknowledgement

We would like to thank the entire personnel of the MoMUPE project and numerous authors of services presented.

References

[1] A. K. Dey. Providing Architectural Support for Building Context-Aware Applications. PhD thesis, College of Computing, Georgia Institute of Technology, 2000.

[2] Browser game Travian. Available: http://www.travian.com/

[3] Overview of Multiplayer Mobile Game Design. Available: http://www.forum.nokia.com/main.html

[4] K. Koskinen, R. Suomela. "Rapid Prototyping of Context-Aware Games", Proceedings of the 2nd International Conference on Intelligent Environments (IE06), Jul 2006

[5] R. Suomela., K. Koskinen, K. Heikkinen. "Rapid Prototyping of Location- Based Games with the Multi-User publishing Environment Application Platform", Proceedings of The IEE International Workshop on Intelligent Environments, Jun 2005, pp.143-151.

[6] R. Suomela, E. Räsänen, A. Koivisto, J. Mattila: "Open-Source Game Development with the Multi-User Publishing Environment (MUPE) Application Platform", Proceedings of the Third International Conference on Entertainment Computing 2004, pp. 308-320

[7] Context-awareness, Available: http://en.wikipedia.org/wiki/Context_awareness

[8] EU-project ePerSpace. Available: http://www.ist-eperspace.org/

[9] Public MUPE services. Available: http://www.mupe.net/applications/

[10] Animal Crossing. Available: http://en.wikipedia.org/wiki/Animal_Crossing

[11] Lautamäki, A. Heiska, T. Mikkonen, R. Suomela, "Supporting Mobile Online Multiuser Service Development", Proceedings of the 3rd IET International Conference on Intelligent Environments, 2007, pp.197-204.

[12] Pyy. M, Heikkinen. K, Porras. J, “Automating Context-Aware Service Development”, Proceedings of Context-Aware Proactive Systems 2007 (CAPS'07), Jun 2007.
[13] Metrics Eclipse plug-in, Available: http://metrics.sourceforge.net/
[14] S.L.Pfleeger, N.E. Fenton: Software Metrics: A Rigorous and Practical Approach, 1997, ISBN 0534956009
[15] Soyatec eUML2 Eclipse plug-in. Available: http://www.soyatec.com/euml2/installation/

PUBLICATION III

Janne Lautamäki and Riku Suomela. "Using player proximity in mobile multiplayer games: experiences from Sandman." In Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, pp. 248-251. ACM, 2008.

Using Player Proximity in Mobile Multiplayer Games – Experiences from Sandman

Janne Lautamäki
Tampere University of Technology
P.O.BOX 553
FI-33101 Tampere
+358405198898
janne.lautamaki@tut.fi

Riku Suomela
Nokia Business Center
P.O.BOX 100
FIN-33720 Tampere
+358504835717
Riku.suomela@nokia.com

ABSTRACT

In addition to using Bluetooth as a communication channel, it can be used to discover other devices nearby. In games, such information can be used in many ways, such as to group people or direct player to player interaction for instance. In this paper, we explore the possibility to use social proximity in multiplayer gaming using mobile phones. As an example, we describe Sandman, which is a context-aware game built on the Multi-User Publishing Environment (MUPE) platform. The game is available for mobile phones with an access to the internet, Bluetooth, and Java MIDP 2.0

General Terms

Experimentation, Human Factors.

Keywords

Context-Aware, pervasive game, Bluetooth, MUPE, multiplayer mobile games, proximity.

1. INTRODUCTION

Context-awareness refers to services' ability to react to changes in the environment [1]. A good example of context-awareness is proximity; many mobile phones have Bluetooth capability, which can be used to detect nearby devices by scanning for their BTUID (Bluetooth Unique Identifier) [2].

In this paper, we present a sample game that uses social proximity in player-to-player interaction. It is called Sandman, and it is based on people monitoring other players in the real world. The game is almost like tag game with digital scoring.

The rest of this paper is structured as follows. Section 2 presents background. Section 3 presents our example proximity game. Section 4 discusses things that effect how the game is played. In Section 5 we present the actual test games and results and Section 6 finally concludes this paper.

Advances in Computer Entertainment Technology 2008, Yokohama, Japan.

2. Background

Many context-aware games use the real world as the gaming arena [3]. In player proximity based games, the device monitors the area around player and tries to find other devices nearby. In our case we used Bluetooth to sense proximity and MUPE (Multi-User Publishing Environment) [4] for implementing the game.

In our game the proximity has two states: another person is in the proximity, or is not. In addition to the two states, the transition from one state to another is important, that is, enter proximity, and leave proximity. With a more complex sensor than Bluetooth it would be possible to detect the distance also, but we are only focusing on discrete stages, and their transitions.

MUPE is an Open Source application platform for creating mobile multi-user context-aware applications. It has a client-server architecture, where the end-user uses client to connect their mobile devices to the MUPE server. External information can be added to MUPE server with context components. Client has an access to the most information supplied by the mobile phone; it can gain an access to Bluetooth, GPS, camera APIs, etc. The MUPE core connects the different parts of MUPE as illustrated in Fig. 1.

In comparison to traditional applications, MUPE applications are deployed differently. MUPE only requires a single client install, after which the user is free to browse new services, and use them with out any extra installations. Client-side of MUPE is build on top of J2ME, and therefore MUPE services are available for the majority of mobile phones.

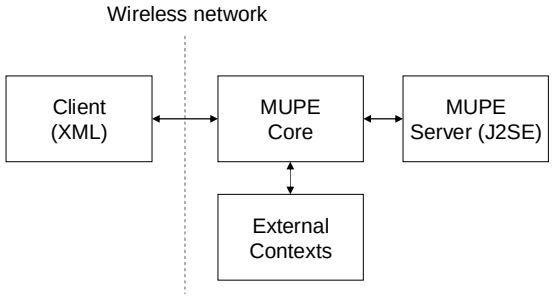

Figure 1. The MUPE application platform structure

3. Example game: Sandman

We aim at using the real world as the gaming arena. In previous attempt, Assassin [5], the players look at other players in the real world, and switch to the mobile phone UI only to take photos of other players. In our sample game Sandman, the players look for other players, and only a single button press is required. With games like these, the players are most of the time focusing on real world, and only occasionally focus on the digital device for short periods of time. Sandman uses the social proximity as the main form of game mechanics. Social proximity is very important in many children games, such as tag, hide and seek, police and robber.

In Sandman, every player acts as a Sandman who has the power to put others to sleep. Players are divided into two teams. The objective of a team is to put all the players in the opposing team to sleep. The state machine representing the different states for the player in the game is given in fig. 2. Sprinkling the sand into air (A and E) causes all the players nearby to fall asleep (C). Also the sandman who sprinkled the sand is affected, if he has not used stimulants to stay awake (B). Players can protect themselves by drinking coffee (D). If a player is drinking coffee while sleeping sand is sprinkled then caffeine protects him from falling asleep (E). Only limited number of coffee doses is available for a player. One dose effects one minute and counters all the sleeping effects.

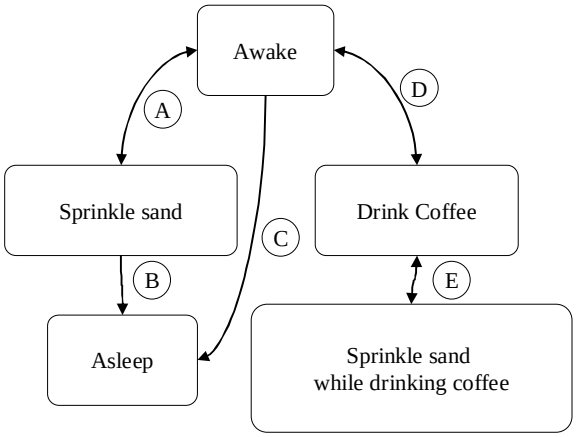

Figure 2. State machine of the game

When sand is sprinkling the phone scans all the nearby Bluetooth devices and sends their BTUIDs forward to the MUPE server. If the BTUID is recognized then the related player falls asleep. Bluetooth is only used for detecting nearby players; the GPRS connection is used to connect the client to the server.

Our tests showed that using BTUID detection is slow. The total time the scanning takes is related to the number of nearby devices. In average it takes about twenty seconds to collect all the BTUIDs nearby. Hence, it is not fast to put another player to sleep. We do not consider this a major problem; it might even make playing the game more interesting. Those seconds give the players a chance to try to escape if a situation is noticed. Running does not always help, since some of the BTUIDs are normally founded fast.

To summarize, the players are trying to minimize the group members and maximize the opponents in the "in proximity" state. The movement in the real world is reflected in the transitions. The simple idea of the game eases the design of a simple and intuitive user interface. In the game room a player has two actions: "Press to Attack" i.e. throw sand and "Press to Defend" i.e. drink coffee, which are enough to play (fig. 3). Lots of other little things are available in UI for making playing more interesting but the game itself is all about pressing two buttons and monitoring actions of the other players in the real world.

Rules should be considered before the session starts. They depend on players and playing area and, obviously change from session to session. The gaming area should be limited. The distance between people during the game has a great effect on how the game is played and how long it lasts. If the people who start the game are near each others, the tension is higher as the players are all the time in the border of the transition between "in proximity" and "out proximity". If the players are more apart there is no possibility for entering the proximity range.

The service itself can not provide strict rules because the game takes place in real world and the device only acts as a judge whether the player successes when trying to sprinkle sleeping sand. In bigger games something like armbands are mandatory for recognizing team mates.

4. Players and Environment

The two key aspects of Sandman are the players and the environment. Environment is a level of the game, and players are its characters. The game session is very different if it is played in large outdoor spaces with limited cover, compared to indoor game, where the players do not need to have visual contact at all during the game. Bluetooth reaches through walls so sometimes a player does not see who sprinkled the sleeping sand.

Lots of possible strategies are available. One strategy is to avoid other players and wait till most of them have fallen to asleep. Hiding player and his team mates win the session if other players play well and the player succeeds in hiding. Other strategy would be that a player actively tries to find a big concentration of other team players and then puts all of them to sleep. In this case, one quickly runs out of coffee and falls asleep soon but with help of his team mates, he can still win. All players rely entirely on their own team to win.

We hypothesis, that the environment has a huge effect on how the game is played. It gives lots of ideas to people what can be done. If we are gaming inside then it is natural that players act as they are supposed to act inside buildings. When playing outside at park players are more likely to run around and chase each others.

Another big factor is the effect of age. With young children the game can resemble a lot like the tag game. If the players are little older and are working at a same office, they can play the session while working. Session can be started at morning and can be played when ever there is enough time or convenient situations. Simple games like these can make the morning coffee break more interesting, and still not require the players to spend time playing the game – it is all about timing.

5. Test results

We have tested the game in different occasions with different people. Designing the game has been highly iterative process. At early tests we had lots of cruder graphics and not so many functionalities and game has slowly evolved to its current form. The main point of these test sessions has been to gain ideas and get some feedback.

In focus group session we had six students aged around 25. They tried out couple of applications and one of applications was an early prototype of Sandman. The interview was organized in a laboratory and it was recorded on a VCR. The players had no chance to escape from laboratory room and so they could not really chase each other. Instead that they tested the user interfaces and discussed about the idea of game.

The participants liked the game and gave some valuable comments for future work. It was also noticed that game was very easy to play. They also gave impression that game would suit better for little younger players and proposed new play modes. They also questioned why the player has to always fall asleep. We decided to introduce the coffee as a game element.

The participants also suggested that the non-player BTUIDs could somehow affect to the game. That kind of item is for example people not playing the game or other Bluetooth device, such as printer. It would be possible to for example use them as a coffee automates to refill ones thermos bottle or something like that.

Finally the game was tested with two test groups. First of them consisted of university students, whose ages varied around 25 years. The second group consisted of 11 years old elementary school students. In first group there were eight players and in second there were eleven. All test subjects were males. With elementary school students all phones were Nokia N95 and with university students there were a variety of Nokia N95, N70 and 7610 phones.

Many things varied between the tests. The gaming area and the age group were different. Students played in sauna compartment of Tampere University of Technology and children played outside, at the park nearby their elementary school. In the first test the players were asked to stay inside the building and in the second one they were not allowed leave from park.

Both of the tests took little over one hour. The first thing was to explain rules, then couple of test games were played in a single room, so that everyone has change to get used with user interface. After that the real games were played. Each game session took about ten minutes. Finally there was debriefing session with feedback forms and free conversation.

Test game with university students was organized in conjunction with a gaming evening of a game development course and there were a lot of games to be tested. Space was also very limited and it only consisted of couple of rooms and long corridors. Some of the players just sit and tried to play without moving. Winner was the player whose coffee lasted the longest. There was no time to plan strategies or other thing like that, because player had to constantly monitor if effect of coffee has ended and then drink a new one. Some players decided to keep distance from other players. When they come back, other players had run out of coffee and players hiding in corridor won the session because they still had coffee left.

Younger test group was much more willing to run and chase each others. It was lots more fun to test game with younger test group. Problems were opposite than with older test group. When explaining the rules there was constant swarm of questions and the participants were really enthusiastic about starting the game session. When game started they run wildly around the park pointing each others with mobile phones and shouting things like “I am sprinkling sand at you”, “I am drinking coffee”. With them it was more like a random thing who actually won. There were no strategies, just a constant chasing.

Figure 3. Example screenshots of the game UI

Even though the sessions were totally different, we can draw some conclusions on. As expected, the school children were very prone to run around whereas it is more of an effort as people grow up. The open environment was a totally different environment – where people had so many visible threats that the strategies were bound to be chaotic. On the other hand, the game session indoors was too small compared to the Bluetooth range, so the level was not optimal for game play. It would be nice to try to but things other way around.

In Table 1, we present some of the results from the questionnaire we made for both of the test groups. Answers are in a scale from one to five. The key finding from questionnaire was that both age groups told that game was fun to play. But in question "Would you like to play it again" only children told that they would like to play again. We are not drawing any conclusion why this is, as the environment, age or the game itself to name a few had an effect on the experience. Based on questionnaire it also seemed that children also tried more actively to get into situations where sprinkling the sand was possible while students played more passive way. This is probably partly due the environment, but younger group's habit to play games probably also influenced to the results. Children answered more positively to all questions. Younger players tend to answer 0.5 points more positive. From the 29 questions asked only in four the difference between groups was zero.

Table I

Aspect	11 years (11 participants)	25 years (8 participants)
Playing the game was fun.	5	4
I would like to play it again.	5	2
I would like to play other games with same battle system.	4,5	3
I actively tried to get situations, where I could sprinkle sand.	4	1

Commenting was voluntary and most comments came from elementary school students. There were comments like: "Game was totally great! It would be fun to play it with my own mobile phone too.", "Game should have faster pace.", "Coffee was too effective, texts were fun, it was hard to put people asleep.", "Simple but boring.", "Game was nice, but probably a bit too simple.". Most of the comments were positive. Negative comments mostly claimed that game was too simple.

We consider that with the normal tag game the results would be same. Basically we learned that digital scoring and mobile devices did not affect so much to adults, but it certainly made children run. Playing tag game is more common for kids than for adults as adults tend to prefer sports. It would be interesting to try to disguise this game to look more like sport than playing and then try to test if adults would like it more. A possible setup would be e.g. competitive orienteering where people would be bound to meet up in certain locations.

To test Bluetooth as proximity detector, we took different models of Nokia phones, and placed them in single room. There were seven phones and 25 tests for each, so the total number of devices to be found was 150. There were slight problems with the platform but no problems with finding BTUIDs. Tests were made in a single room, so it is possible that there are asymmetries between founding the devices if distance is greater. A bigger problem is the time the scan takes. Depending on the device and amount of other devices nearby, scanning can last from a couple of seconds up to almost a minute. It is not always the case that the oldest phone is the slowest; it is more like a random thing. Games that depend on Bluetooth as a proximity detector is definitely unfair to some of the players.

6. CONCLUSION

In this paper, we presented a case example how the player proximity in the real world can be used in mobile multiplayer games. A sample game implemented with MUPE works in most phones. Bluetooth is used to detect other players nearby. The game was tested with several groups. These groups had a totally different kind of a game experience, which was due to both the age and the environment they played in. Different kind of strategies and game situations were described. Moreover, an extended discussion was given to analyze the game.

Implementing other game modes from the first person shooter genre would make more variance to game play. Other methods to sense proximity like GPS and WI-FI could also be tried out. It would be interesting to test Sandman in events with high density and number of people.

Also some platform-related issues were discovered. With MUPE there is no way to invite other users to a certain service or to create groups of friends. If MUPE wants to be "the mobile Facebook", then at least these features should be included.

To gain a deeper understanding of the differences, further study is needed on the effect of the environment as a game level, and the effect of the age and the group dynamics.

REFERENCES

[1] Dey, A. K., 2000. Providing Architectural Support for Building Context-Aware Applications. PhD thesis, College of Computing, Georgia Institute of Technology.

[2] Bray, J., and Sturman, C., 2001, Bluetooth: connect without cables, Prentice Hall Inc, Upper Saddle River.

[3] Schilit, B., Adams, N. & Want, R., 1994. Context-Aware Computing Applications. IEEE Workshop on Mobile Computing Systems and Applications, In Proceedings of the Workshop on Mobile Computing Systems and Applications.

[4] Suomela, R., Räsänen, E., Koivisto, A., Mattila, J., 2004. Open-Source Game Development with the Multi-User Publishing Environment (MUPE) Application Platform. Proceedings of the Third International Conference on Entertainment Computing 2004, 308-320, Lecture Notes in Computer Science 3166 Springer.

[5] Suomela R., and Koivisto A., 2006. My photos are my bullets - using camera as the primary means of player-to-player interaction in a mobile multiplayer game. 5th International Conference on Entertainment Computing – ICEC, Lecture Notes in Computer Science 4161 Springer.

PUBLICATION IV

Janne Kuuskeri, Janne Lautamäki, and Tommi Mikkonen. "Peer-to-peer collaboration in the lively kernel." In Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 812-817. ACM, 2010.

Peer-to-Peer Collaboration in the Lively Kernel

Janne Kuuskeri
Department of Software Systems
Tampere University of Technology
P.O. Box 553
FI-33101 Tampere, Finland
janne.kuuskeri@tut.fi

Janne Lautamäki
Department of Software Systems
Tampere University of Technology
P.O. Box 553
FI-33101 Tampere, Finland
janne.lautamaki@tut.fi

Tommi Mikkonen
Department of Software Systems
Tampere University of Technology
P.O. Box 553
FI-33101 Tampere, Finland
tommi.mikkonen@tut.fi

ABSTRACT
With the increasing popularity of the World Wide Web, end-user applications are moving from the desktop to the browser. More and more applications that we have come to know as desktop applications are now making their way into the web. This has made online collaboration a key aspect for many applications. Collaborative applications like Facebook, Flickr and Google Docs are just an early hint of how we can benefit from users being able to share data. Still, collaborative features are not easy to implement in web applications. Most web programming environments aim at easy user interface creation and persistence, but not for online collaboration and pushing data from one user to another. The Lively Kernel is a highly dynamic web programming platform and runtime environment developed at Sun Microsystems Laboratories. By utilizing the Lively Kernel platform, it is easy to implement desktop like applications for the web using JavaScript. In this paper we summarize the experiences from adding peer-to-peer collaboration into the Lively Kernel. These additions include persistent data storage, communication channels between users and user identification.

Categories and Subject Descriptors
D.2.6 [**Software engineering**]: Programming Environments

General Terms
Design, Experimentation

Keywords
Collaboration, JavaScript, Lively Kernel, Web Application

1. INTRODUCTION
During the recent years, web applications have become more and more popular. Today, it is not unusual for complex applications to use the web as their only platform, with no traditional user interface for the desktop. Good examples of these are applications like Flickr, Picnik and Google Docs. We believe that the transition from the desktop to the web has only started, and the variety and importance of web applications is constantly rising. Web applications are easy to adopt since they need neither installation nor updating. They are also easy and cheap to publish and maintain; there is no need for intermediates like shops or distributors [1]. Furthermore, in comparison to conventional desktop applications, web applications have a whole new set of features available, like online collaboration, user created content, shared data, and distributed workspace.

Many web applications are a fragmented mixture of different kinds of technologies like AJAX, HTML, CSS, DOM and JSP [2]. Moreover, most of them use JavaScript [3] to add client side functionality and to create more dynamic web pages. This easily leads to applications that rely on a number of technologies, and consequently the code base may become very difficult to maintain. To overcome such obstacles, the Lively Kernel is a platform for developing and hosting client side applications that are implemented using JavaScript only [4]. The Lively Kernel provides a rich set of user interface components and an MVC model for event handling thus making the creation of complex web applications much easier.

As the web gains more and more central role in computing, it is only natural that applications move into the web as well. The Lively Kernel helps in this transition by making the browser a platform for applications rather than just being a document viewer. However, in its current state the Lively Kernel does not take full advantage of capabilities of the web. In the situation where all users are connected to a single server, it only seems logical to assume that users can somehow collaborate with each other. One good example of a truly collaborative environment is the recently introduced Google Wave (*http://wave.google.com*), in which users are able to send messages and share data in real time.

In this paper we present some new features for the Lively Kernel in order to make it a collaborative platform, where peer-to-peer communication is possible between different Lively Kernel instances running in different browsers. To add peer-to-peer collaboration to the Lively Kernel, we have extended the platform with a variety of new features and applications such as user identification, friend list, chatting, two player gaming and data sharing. Furthermore, we have implemented a concept of remote wormholes into the platform. These wormholes connect two remote Lively Kernel instances, and users can use them for sharing objects, including even ones that include runnable code. With these extensions developers are able to implement truly

SAC'10, March 22–26, 2010, Sierre, Switzerland.

collaborative applications and users are able to interact with each other in real time.

The rest of the paper is structured as follows. In Section 2 we describe the original Lively Kernel and its features. In Section 3 we provide an overview of features we have created and discuss more about technical details. In Section 4, we draw some final conclusions.

2. THE LIVELY KERNEL

Currently, most web applications – excluding some rare exceptions – are separate instances that do not interact with each other. In collaborative applications, the server commonly plays a major role by providing a single state that is shared. This is partly a result of lacking mechanisms and interfaces to carry out the interaction between different clients in real time and partly because it is just impossible. In contrast, desktop applications have multiple means for communication; they may share files or communicate directly via various messaging protocols, whose features vary considerably.

In the Lively Kernel, the user interface already resembles to the ones we are familiar with from the desktop. Therefore, it is only natural to provide the collaborative services resembling those of conventional desktops to Lively Kernel applications as well.

2.1 Origins of the Lively Kernel

The Lively Kernel (*http://research.sun.com/projects/lively/*) by Sun Microsystems Laboratories is an interactive web application platform that has been written entirely in JavaScript. Being a web application itself, users of the Lively Kernel do not need to install anything in order to use it. The only thing that is required is a compatible web browser. The list of compatible browsers includes the latest versions of Firefox, Safari, Opera and, Google Chrome. The Lively Kernel can be hosted on a web server for others to use or it can reside on user's local machine in which case no web server is required. A screenshot of the Lively Kernel is provided in Figure 1.

For application developers the Lively Kernel provides a rich set of user interface widgets, an event model and other useful features such as modules, classes and networking. Applications are implemented purely in JavaScript. There may be several applications running simultaneously inside the Lively Kernel, and they all remain active while another application is used. However, due to the restrictions of the client-side JavaScript environment, the applications share the same virtual machine that executes them in a single process, and parallel executions have been executed in terms of the JavaScript eventing system at the level of implementation.

Due to the higher level of abstraction, writing applications for the Lively Kernel is easier than writing them directly for the web browser. In fact, implementing applications using the Lively Kernel resembles the traditional way of implementing desktop applications. With the Lively Kernel, developers need not to worry about things commonly associated with using JavaScript inside browsers, like manipulating the Document Object Model (DOM) directly or defining cascading style sheets (CSS). With the

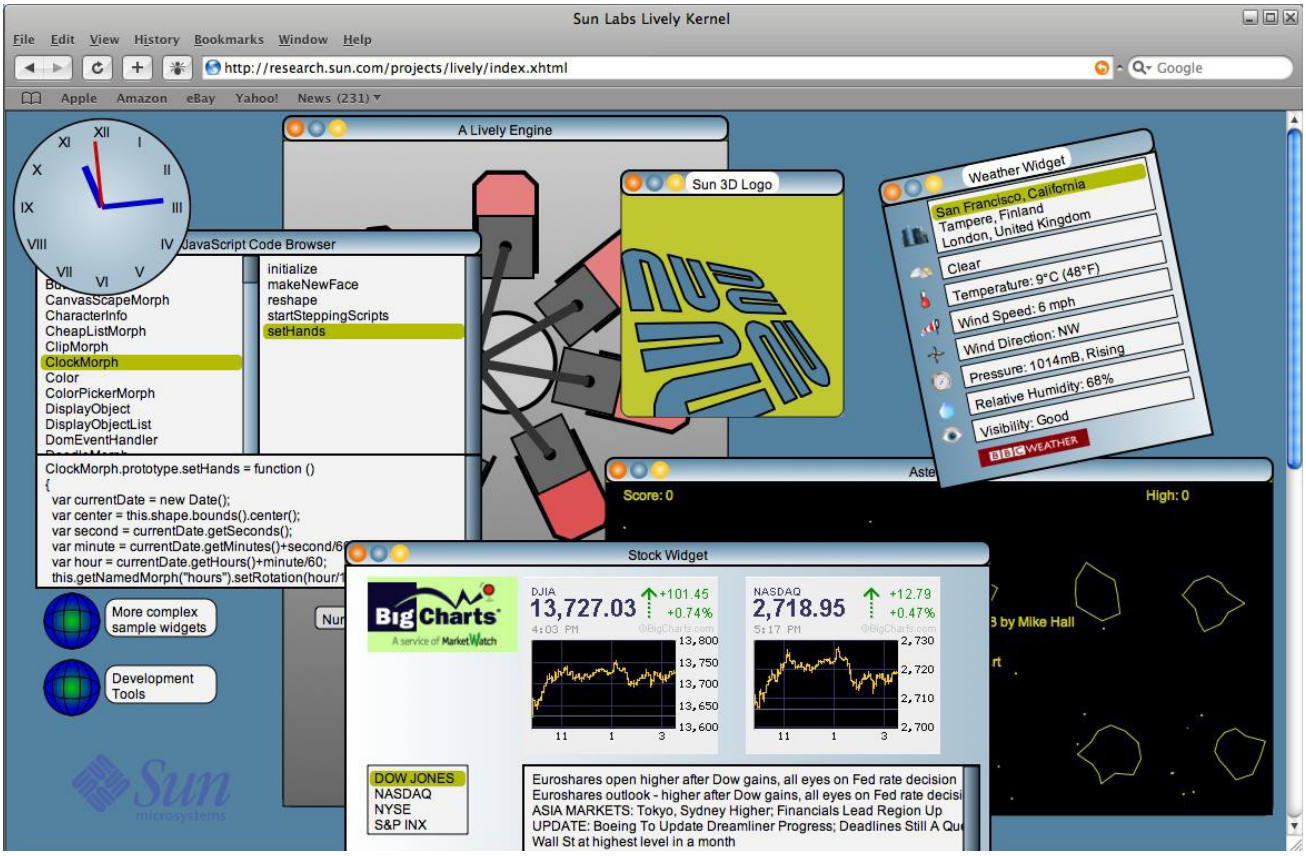

Figure 1. The Lively Kernel running in Safari browser [4]

Lively Kernel, developers only have to know how to write JavaScript and familiarize themselves with the Lively Kernel APIs.

The Lively Kernel is based on the *Morphic* user interface framework [5]. Within the framework there are four important main concepts, *World*, *Morph, Wormhole,* and *Hand*. These are introduced in the following:

Worlds. The concept of a world simply refers to a workspace within the platform. This is very similar to a virtual desktop used by many windowing systems. Inside the main world, it is possible that there are so-called sub-worlds, which can be accessed from the main world.

Morphs. Similarly to other Morphic implementations, Lively Kernel morphs are simply graphical objects. All morphs reside in a world and they can be seen as objects, applications or widgets. In Figure 1 we can see several morphs and a world as a canvas for them.

Wormholes. Inside a Lively Kernel instance, there can be only one visible world at a time, but the world can contain links to other worlds. These links are called wormholes. In Figure 1, two blue circular wormholes can be seen at the bottom left corner. These wormholes act as links between local worlds running inside a single instance of the Lively Kernel – in other words a single web page. The user can use the wormholes for dragging objects from one world to another. Wormhole icons can also be clicked in order to move to another world, each of which can contain a different set of morphs.

Hands. The morphs in a world are manipulated using a hand, which is a generalization of the cursor. In terms of the underlying implementation, the hand is also a morph that has all the properties of a regular morph as well.

2.2 Collaboration through wormholes

We have amended the Lively Kernel with persistence and peer to peer collaboration. They have been implemented as a part of the platform. With these additions we were able to make the Lively Kernel more appealing for both, the end users and the developers and to turn the Lively Kernel into a collaborative platform in which users can share data, and thus gain the full benefits of being online.

Furthermore, we have extended the concept of wormholes. Instead of being used inside one browser only, the concept has been extended for sending live morphs between remote users, each having their own worlds. The scheme has been illustrated in Figure 2. In the figure, rectangles represent different Lively Kernel instances belonging to different users. Circles represent the different worlds running inside different Lively Kernel instance. Finally, arcs inside a rectangle represent local wormholes and arcs that cross the boundaries rectangles are the new wormholes that enable connections from one user to another.

Since the scheme connects different users directly, this can considered as a peer-to-peer system. However, due to the restrictions of the browser as a platform, behind the scenes the traffic is actually routed through the server. These restrictions are associated with the current security model in browsers, namely the same origin policy [6], which makes it impossible to bypass the server. Nevertheless, these new kinds of wormholes open a whole new set of possibilities for developers and end users, since live morphs can now be shared with other users and sent back and forth.

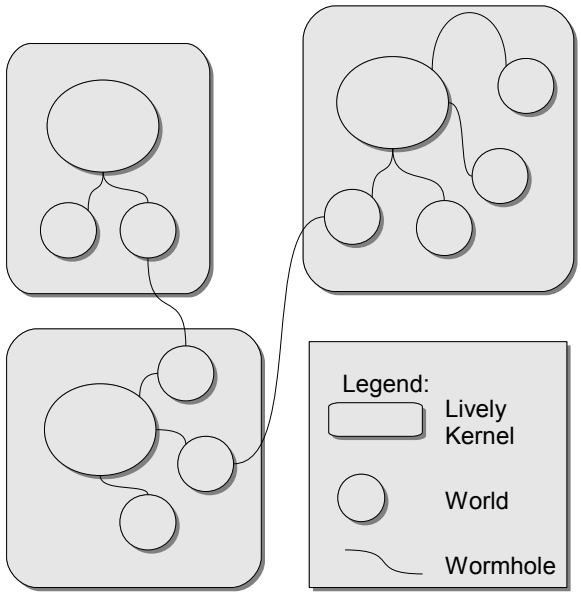

Figure 2. Wormholes between Lively Kernels and worlds

3. IMPLEMENTATION

The implementation we have composed to enable peer-to-peer collaboration can be divided into three main parts; the server side implementation and the related communication channels, the serialization of transferred data and the user interface which enables the use of the collaborative features. Next, we describe how these different components are implemented.

3.1 Amendments to the user interface

For a user of any collaborative application, one of the key requirements is to see who else is online. For computer programs this is fairly straightforward as we could just use some unique string of characters to represent a client. However, for users to be able to identify one another there needs to be some familiar name associated with the user. The most common technique for accomplishing this is to require the users to register and log in to the system before they start using it. Our login screen can be seen in Figure 3. This screen can be used for both to register a new user and to login as an existing user.

In most community based applications and platforms, users are able to form groups of users they want to collaborate with, into their own lists. These lists are generally known as user's *friends* [7]. We have also added friend list to the Lively Kernel. This list can be made visible from the main application manager, and it is automatically updated as users become online.

The application manager and the friend list are presented on the left side of Figures 4 and 5. By clicking on a friend on the list, a new window is opened. This window is called friend panel, and it is presented on the right side of Figure 4. besides enabling chatting, it opens a wormhole between the two users. The wormhole can then be used for sending all kinds of objects that live inside the Lively Kernel, including basic morphs, pictures and applications. Moreover, since morphs can contain program code,

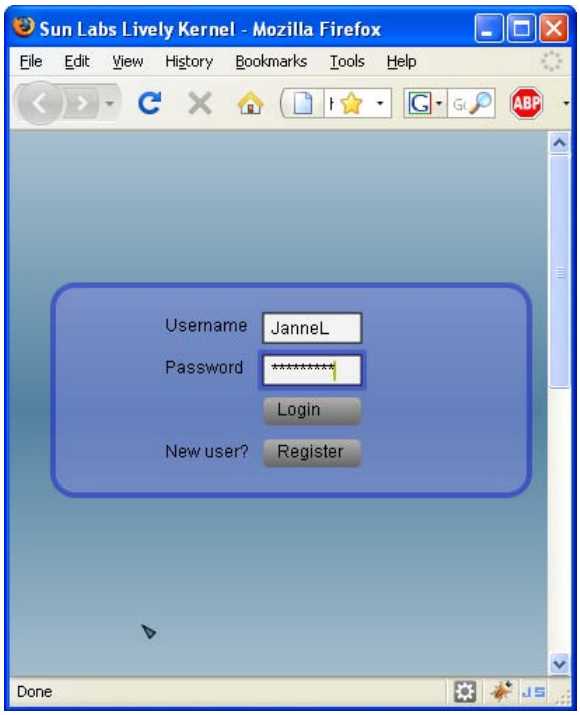

Figure 3. Login screen

also this code can be set through the wormhole from one user to another. Hence, the wormhole essentially acts as a peer-to-peer communication mechanism that is readily available for users.

For example, such wormhole can be used to initiate games or other applications. User could suggest that *"Let's play a game of Dung Beetle"* to a friend and then the game itself could be dragged and dropped on the very same chat screen to initialize the game for the friend too. The Dung Beetle is actually a two player game that we have created to demonstrate the collaborative features (see Figure 5). The idea of the game is simple: many dung beetles, known as rollers, are noted for rolling dung into spherical balls, which are used as a food source or brooding

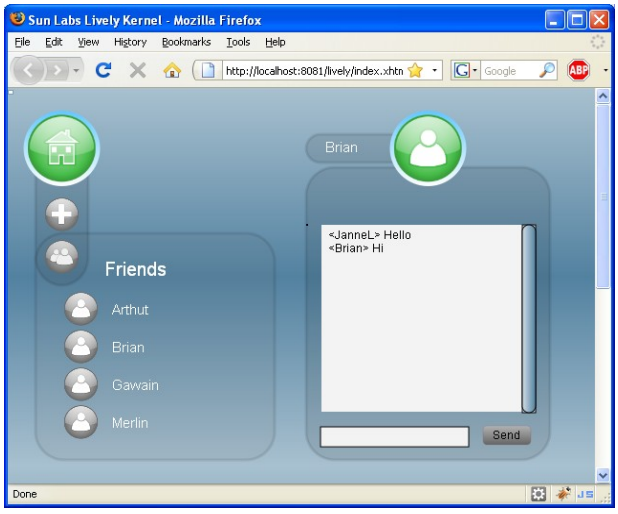

Figure 4. Friends list and the friend panel

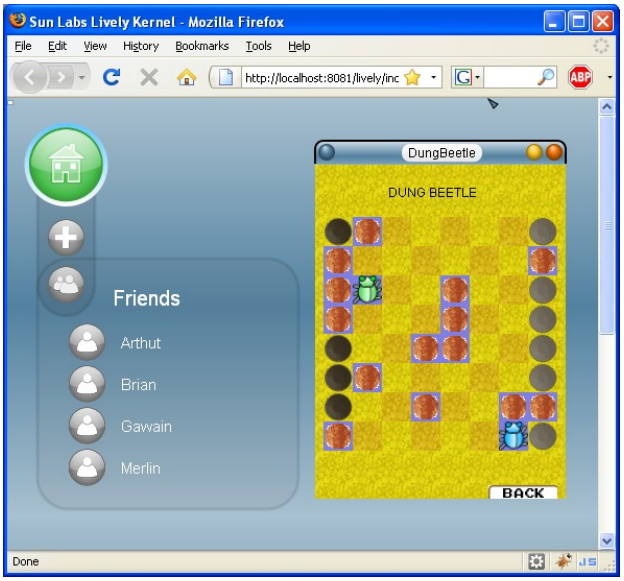

Figure 5. Friends list and Dung Beetle game

chambers. Both players have a dung beetle to control and they are trying to collect falling piles of dung to all of the chambers before the opponent does it first.

3.2 Server side implementation

The server side of our application is also implemented in JavaScript. It uses Rhino JavaScript engine and embeds Jetty web server, as already described in [8]. Essentially it provides Lively Kernel application developers access to server side functions and modules. Client side application may use whichever functionalities provided by the server for example database, file I/O, and networking and then the server side platform dynamically replaces those invocations with RPC stubs upon application loading. Hence, when the Lively Kernel is loaded, the server inspects and instruments the code and sends it to the browser and whenever a "server side" function is called in the browser, an Ajax request is made in the background. This way application developers may implement their web applications in a more natural an concise way and not worry about the networking details. They can just use modules and services provided by the server as if they were local.

In the context of adding collaboration to the Lively Kernel, there are two interesting modules provided by the server: *friends* and *database*. The friends module can be used to send either multicast messages to all users or private messages to a selected user. Messages are delivered asynchronously and received by other Lively Kernel applications via Comet channels. Applications are free to create new channels or register into existing channels. The database module, on the other hand, comprises basic create, read, update and delete (CRUD) operations against the database. With these two modules we are able to turn the Lively Kernel into a fully collaborative platform.

3.3 Serialization

One of the key ideas from the start was to create a new kind of wormhole to send objects and applications from one user to another. Sending an object over HTTP is divided into three steps.

First, the object is serialized into text, then the text is sent to a friend and finally the receiving application deserializes the text back into object. How the object is serialized and what is the actual textual format is a compromise of many different aspects. Different formats may vary from easily readable JSON and XML to some compact binary formats.

In case of the Lively Kernel the serialization should support a tree of objects. However, for JavaScript this creates a problem. Serialization would easily extend to object's prototype and in worst case to the entire top-level scope and everything it refers to. Hence, there should be some limit in how deep the serialization goes into. Sometimes it is intentional to serialize the whole object tree and sometimes it is not. In our case, we want to be able to serialize a JavaScript object and the child-objects it refers to and then deserialize them into a new scope and have all of the references from the deserialized object to prototypes and parent scopes resolved correctly to refer to objects in the new scope. In our study we have identified three different approaches on how to serialize and deserialize objects. Each of the different approaches have their benefits and flaws.

The first and the easiest way is to simply send source code over the web, as shown in Listing 1. With this method, the receiver needs detailed instructions on how to initialize object. Another alternative would be to define a robust standard on initializing incoming objects. Furthermore, in most cases the state of the system must be preserved. Hence all the variables have to be sent together with the object. With this approach the hostile client has a good opportunity to upload whatever code it wishes to other users.

The second approach is to take an object and then serializing it back to source code. This approach has been demonstrated in Listing 2. This piece of code can then be sent to friend and then deserialized again by using *eval()* function that is readily available in JavaScript interpreters. Basically this approach has the same benefits as the first one, but there is no need for extra information, because all the variables are included in the serialized object. Nevertheless the problem associated with hostile clients remains. Other users can send hostile code and it is virtually impossible to create adequate safety mechanisms for the receiver.

There is one more problem with both of the above approaches. Even if all the variables are serialized along with the object, it is still possible that some structures that affect to the object are lost in the serialization process. For example, in the Lively Kernel it is easy to create a new timer for moving your object but if you do not have reference from object to timer then timer does not get serialized with your object. Then, even if the timer is adequately serialized, it still has to be started. The same thing is applicable to mouse and keyboard events, for example.

Difficulties when serializing events results from the structure of the Lively Kernel. Timers and events are not easily accessible from the object itself, because they are not stored inside morphs. In the present implementation, we decided that not to worry about this problem. In the future, an easy workaround would be to add a new method called *refreshEvents()*, which would reinitialize timers and other events after the deserialization process.

Also further problems emerge with the second approach. Most of the serialization algorithms use recursion. However, many of the current JavaScript virtual machines are not designed for deep recursion. While there is no limit for call stack size in the

Listing 1: Source code

```
Object.subclass("Rectangle", {
  documentation: "primitive rectangle",
  initialize: function(x, y, w, h) {
    this.x = x; this.y = y;
    this.width = w; this.height = h;
    return this;
  },
  copy: function() {
    return new Rectangle(this.x, this.y,
                this.width, this.height);
  },
  maxX: function() { return this.x + this.width;},
  maxY: function() { return this.y + this.height;},
  ...
```

Listing 2: Object serialized to source code

```
{
  x : 0, y : 0, width : 60, height : 30,
  constructor : function Rectangle() {
    Class.initializer.apply(this,
                            arguments);
  },
  documentation : 'primitive rectangle',
  initialize : function(x, y, w, h) {
    this.x = x; this.y = y;
    this.width = w; this.height = h;
    return this;
  },
  copy : function() {
    return new Rectangle(this.x, this.y,
                this.width,this.height);
  },
  maxX : function() { return this.x + this.width;},
  maxY : function() { return this.y + this.height;},
  ...
```

definition of the language, the actual maximum size of call stack varies from browser to browser, and it can be as low as 100 [9]. Recursion can of course always be transformed to loop structures by using stack, but this makes code more complicated. Other thing is that there are circular references inside the Lively Kernel and they should be detected. For example, a DOM node has references to its children and these children also have references back to the DOM node. When serializing this kind of structure, it is easy to end up in an infinite loop or an exception. So when serializing it is mandatory to keep track of items that are already visited. One must also keep in mind that there can be multiple references to single object and when serializing it should be serialized only once and all the references should point to that single instance.

Finally, the third approach is to serialize object to some kind of formatted XML-structure and use custom deserializer for deserializing the object. This approach has been sketched in Listing 3. With a custom serializer it is easier to draw the line on what kind of objects can be sent through the wormhole. In our case only the objects whose source code is already available for

Listing 3: Object turned to XML:

```
<g xmlns="http://www.w3.org/2000/svg"
   type="Morph" id="161:Morph"
   transform="translate(- 29, -14)">
  <rect x="0" y="0" width="60" height="30"
        stroke-width="1" stroke="rgb(0,0,0)"
        fill="rgb(0,0,204)"/>
  <field xmlns="" name="origin" family="Point">
    <![CDATA[{"x":0,"y":0}]]>
  </field>
  <field xmlns="" name="fullBounds" family="Rectangle">
    <![CDATA[{"x":0,"y":0,"width":60,
              "height":30}]]>
  </field>
  <field xmlns="" name="scalePoint" family="Point">
    <![CDATA[{"x":1,"y":1}]]>
  </field>
</g>
```

both users, and can therefore be trusted, can be send. With this approach there is still the same event and timer problem as with second alternative. However, also solution to that problem can be the same, and composing a practical implementation is not hard.

4. CONCLUSIONS

In this paper, we presented new features for enabling community aspects into the Sun Labs Lively Kernel platform. We created a simple login mechanism to identify old and new users, and this way demonstrated the database capabilities of the server. We also created a friend list for users to see who is online. The friend list can be used for opening private chats between friends, but it can also act as an extension to wormholes by offering a channel to send all kinds of objects available in the Lively Kernel. We also presented a multiplayer game to demonstrate how communication channel can be used to send actions from one user to another.

Adding online collaboration to the Lively Kernel platform has made it much more appealing to the users and allowed for totally new kinds of applications to be implemented. Sharing data or even applications themselves gave us the true power of online collaboration. Seeing this as a proof of concept, we plan continue our study and apply these same techniques on another platform called Lively for Qt [10].

Lively Kernel version 0.8.5., which we have used as the base for our work, lacks the support for collaborative features. However, the researchers at Sun Microsystems Laboratories have developed the platform further and they have added some new collaborative aspects in it. Those features do not conflict with our additions and with careful work, the two versions could probably be combined. The new version with collaborative features from Sun Microsystems is called Lively Wiki because it has wiki like features; users can contribute to the community by creating their own content and then saving and sharing it for other users to use (*http://livelykernel.sunlabs.com*). However, this requires reloading of the page and thus, is somewhat limited approach to true online collaboration of applications.

We are also planning on implementing remote controlled objects on top of the wormhole concept. In this scenario users would be able to share user interface components via the wormhole. For example, they could share a text editor where both users would be able to write and the changes would immediately be visible to both users. This idea could be extended into worlds themselves. Hence, when the user enters to a world, it would actually be a remote world inside a remote Lively Kernel.

Acknowledgements

This research has been supported by the Academy of Finland (grant 115485).

REFERENCES

[1] O'Reilly, T. *What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software.* Communications & Strategies, No. 1, p. 17, First Quarter 2007.

[2] Paulson, L.D. *Building rich web applications with Ajax.* Computer , vol.38, no.10, pp. 14-17, Oct. 2005.

[3] Crockford, D. *JavaScript: The Good Parts*. O'Reilly Media, 2008.

[4] Taivalsaari, A., Mikkonen, T., Ingalls, D., Palacz, K. *Web Browser as an Application Platform: The Lively Kernel Experience.* Sun Microsystems Laboratories Technical Report TR-2008-175, January 2008.

[5] Maloney, J.H. Morphic: *The Self User Interface Framework.* Self 4.0 Release Documentation, Sun Microsystems Laboratories, 1995.

[6] Ruderman, J. *The same origin policy.* 2001, *http://www.mozilla.org/projects/security/components/same-origin.html*.

[7] Boyd, D.M., Ellison, N.B. *Social Network Sites: Definition, History, and Scholarship.* Journal of Computer-Mediated Communication vol. 13, no. 1, pp. 210-230, 2008.

[8] Kuuskeri, J. and Mikkonen, T. *Partitioning web applications between the server and the client.* Proceedings of the 2009 ACM Symposium on Applied Computing, Honolulu, Hawaii, 2009

[9] *Maximum Call Stack Size in Modern Day Browsers. http://novemberborn.net/javascript/callstack-size*

[10] Mikkonen, T., Taivalsaari, A. and Terho, M. *Lively for Qt: A Platform for Mobile Web Applications*. In Proceedings of the Sixth ACM Mobility Conference, Nice, France, September 2-4, 2009.

PUBLICATION V

Timo Aho, Adnan Ashraf, Marc Englund, Joni Katajamäki, Johannes Koskinen, Janne Lautamäki, Antti Nieminen, Ivan Porres, and Ilkka Turunen. "Designing IDE as a Service." Communications of Cloud Software 1, no. 1, p. 10, 2011.

Designing IDE as a Service

Timo Aho* Adnan Ashraf† Marc Englund‡ Joni Katajamäki** Johannes Koskinen*
Janne Lautamäki* Antti Nieminen* Ivan Porres† Ilkka Turunen**

Tampere University of Technology*
P.O.Box 553, FI-33720 Tampere, Finland
firstname.surname@tut.fi, except antti.h.nieminen@tut.fi

Åbo Akademi University†
Joukahainengatan 3-5, FIN-20520 Turku, Finland
firstname.surname@abo.fi

Vaadin Ltd.‡
Ruukinkatu 2-4, FI-20540 Turku, Finland
marc.englund@vaadin.com

JAMK University of Applied Sciences**
Rajakatu 35, FI-40200 Jyväskylä, Finland
firstname.surname@jamk.fi

Abstract

While the popularity of web applications is growing, most of the software is still developed using desktop tools. Nevertheless, a browser-based development environment could offer a multitude of advantages. For example, only an up-to-date web browser is needed and therefore the user has no need to carry out complex tool installation and update procedures. Furthermore, publishing the applications to the web is easy to offer as a feature of an IDE, and since the users are already connected via server, collaborative features are easier to implement. For beginning businesses, effortless publishing offers new possibilities.

In this paper, we present Arvue, a browser-based tool that enables simple development and publishing of web applications. Arvue applications are created on the browser using a user interface designer and an integrated code editor. The applications are stored in a version control system provided by Arvue and they can easily be published to the cloud. Publishing the user-created applications may impose resource usage and security issues, which are also addressed in the paper.

Keywords: Cloud, Integrated development environment

1 Introduction

One of the main trends of software engineering is that the applications are moving from desktop to the web browser. The trend has multiple benefits: Web applications are available globally and can be accessed using any up-to-date web browser. They are also easy and inexpensive to maintain. Updates are distributed automatically just by updating the application on the server. The web, and especially the usage of the cloud [1], makes it possible for anybody with modest software development skills to create applications and then with small effort and relatively low costs to offer it to the global market.

Despite the success of the web as an application platform, the vast majority of software is developed using a desktop based integrated development environments (IDEs). However, an IDE inside a browser offers several benefits. The developer does not need to worry about installing, configuring or updating the environment as the latest version available on the cloud is automatically used. Along with an IDE, there can be additional developing tools such as a development server or a test harness. Since the developers are connected to the cloud, the created projects can easily be stored in the cloud. In addition, it is possible to implement a set of communication, collaboration and project management tools. Furthermore, it reduces errors if we develop and test applications in the same environment as the actual hosting is done.

In this paper, we describe Arvue[1], that provides the IDE and the hosting as a service. The most visible part of Arvue is a browser-based development environment. The environment contains a visual user interface (UI) designer and a code editor along with an easy way to preview and publish applications. The applications are stored in a version control system integrated with the IDE. We also identify the potential risks related to user-created applications and give our solutions to monitoring and controlling the resource usage of applications as well as to addressing the security concerns.

The paper is organized as follows. In the next section, we give an overview of the technologies used in our implementation. Section 3 discusses the most relevant related work. In Section 4, the overall view of our architecture is presented, as well as more detailed descriptions of the components. Finally, in Section 5 we draw some concluding remarks and discuss possibilities for future work.

2 Background

As already mentioned in the introduction, the benefits of hosting applications in the cloud are evident [1]. The philosophy of cloud computing is to see applications as a service instead of a product. In the cloud, the resources behind the applications can be scaled up and down according to one's needs without owning heavy server infrastructure. In an IDE and hosting service like Arvue, the scaling is naturally a very desirable feature because of unpredictable and changing needs. When selecting the cloud provider, the most typical commercial alternatives are Amazon Web Services[2] and Google App Engine[3]. It is also possible to build cloud of your own by using open-source cloud computing frameworks like OpenNebula[4] and Eucalyptus[5].

A remarkable part of web applications is based on Java programming language. Especially Java Servlet [9] is a fundamental building block for Java based web applications and is used in huge number of tools and frameworks. In this work, for implementation, we use Vaadin [4], that is an open-source framework for developing Java Servlet based web applications. For client side features, Vaadin framework extensively relies on the facilities of Google Web Toolkit[6] (GWT). GWT is an open-source development system that allows the developer to write applications in Java and then compile the source code to JavaScript which can be run on all browsers. When using Vaadin, the developer can implement server side application using the full power and flexibility of Java and Vaadin automatically takes care of the client side.

When using web servers like Apache Tomcat[7] or Jetty[8], all the served web applications are located

[1] Available at http://www.arvue.com.
[2] http://aws.amazon.com
[3] http://code.google.com/appengine
[4] http://www.opennebula.org
[5] http://www.eucalyptus.com
[6] http://code.google.com/webtoolkit
[7] http://tomcat.apache.org
[8] http://eclipse.org/jetty

on the same Java virtual machine. However, Java lacks many important supportive features to run multiple applications on a single Java virtual machine. To patch this up, a widely adapted OSGi[9] specification [17] has been published. OSGi introduces the dynamic component model which allows, e.g., updating, installing and removing components on-the-fly in a single Java virtual machine [6, Section 15.1].

Vaadin framework is compatible with OSGi. As the framework can be configured as an OSGi bundle, it is possible for other OSGi bundles to use it directly. In this way, Vaadin web applications can use the Vaadin framework bundle like a dynamic library resulting in very lightweight applications. Thus, the typical size of the Vaadin application bundle file in OSGi is less than 100 kilobytes, and the starting up of an application is a light and fast operation.

3 Related Work

Many web-based IDEs are released during the past few years. Their type varies from simple programming language tutorials to complete solutions for developing and deploying web applications. The scientific literature mostly seems to concentrate on the collaborative side of IDEs [18, 3, 11, 7] or on solving the technical challenges [19]. On the other hand, some of the web IDEs, such as jsFiddle[10] concentrate only on the client side development with, e.g., JavaScript, HTML, and CSS. In addition, for example JavaWIDE[11] [5, 10] allows the development of Java applets running on the client browser. Nevertheless, some IDEs also support server side development. Because our Arvue is designed for creating software with a thin client and for doing most of the computation in a cloud, we here limit to the IDEs from this category. In addition, we further concentrate on the IDEs that offer some kind of deployment and hosting service for the created applications.

A popular way to include server side functionality in a web application is NodeJS[12]. It is an engine which allows the developer to write the server side code with JavaScript. Some web IDEs that incorporate NodeJS development include Cloud9[13] and AkShell[14]. Both of them offer GitHub[15] integration and use Ace[16] as their client side editor component. In addition, web IDEs offer various other technologies for web application development. For example, eXo Cloud IDE[17] supports a variety of solutions like Java Spring and REST services and includes the wide selection of languages include HTML, JavaScript, PHP, Java, Groovy, and Ruby. For these languages, it offers development aids like automatic code completion.

On the other hand, Coderun[18] is a web IDE for developing C# applications. It allows debugging applications while they are in execution. The client side code is generated using Sharpkit[19] — a C# to JavaScript translator similar to GWT for Java. Finally, a web IDE worth mentioning is Kodingen[20]. Its goal is to provide an easy way to deploy, develop and host web applications using various languages and frameworks. It supports a wide selection of editors and source code control tools.

[9] OSGi originally stood for Open Services Gateway initiative framework. The longer form is at present seldom used because the specification has moved beyond the original focus.
[10] http://jsfiddle.net
[11] http://www.javawide.org
[12] http://nodejs.org
[13] http://cloud9ide.com
[14] http://www.akshell.com
[15] http://github.com
[16] http://ace.ajax.org
[17] http://cloud-ide.com
[18] http://coderun.com
[19] http://sharpkit.net
[20] http://kodingen.com

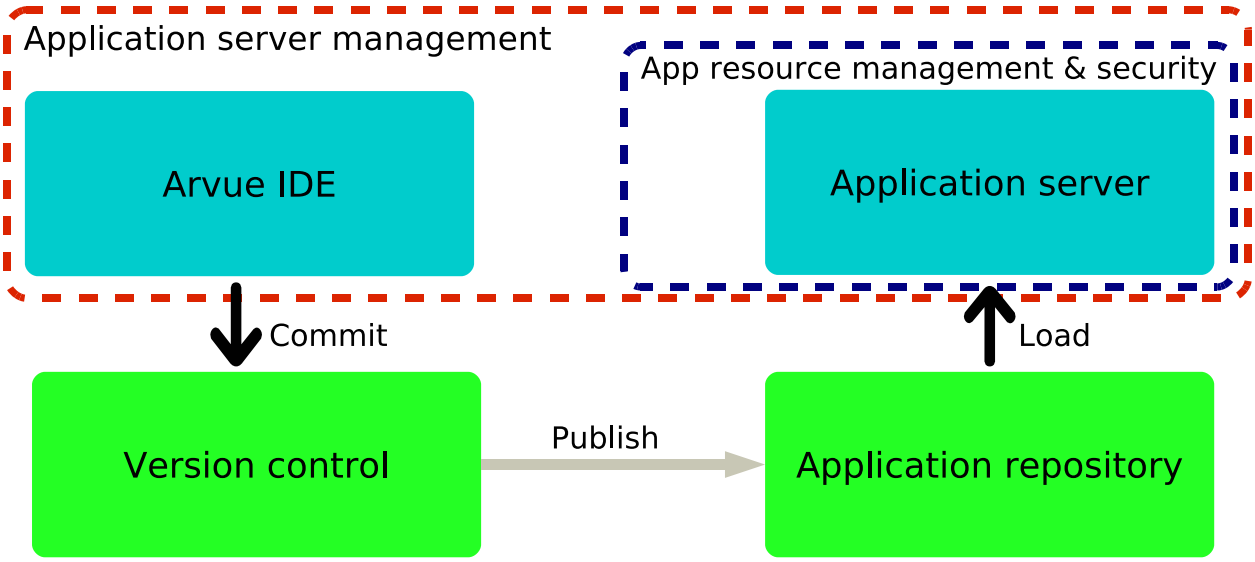

Figure 1. Overall view of Arvue architecture

The mentioned IDEs differ from our solution especially in their basic philosophy. Our intention is to present a useful editor for a single task: creating and publishing small Vaadin based web applications. Because of our narrow application area, we are able to present much more comprehensive process tools than is possible with a general use IDE. As a concrete example, our IDE contains the graphical editor for implementing the application UI and supports only single framework.We will cover the issue more in detail in the following sections.

4 Architecture

In this section, we describe the Arvue architecture and components. The overall architecture of Arvue components is presented in Figure 1. Arvue can be divided roughly in four different components:

- Arvue IDE, both graphical UI and collaborative code editor (CoRED) [11], for creating applications. Components are presented in Section 4.1.
- Version controlling services component covered in Section 4.2.
- Arvue auto scaling framework for hosting applications is described in Section 4.3.
- Monitoring resource usages and security of the applications is covered in Section 4.4.

Arvue IDE runs on a dedicated Jetty WebServer as there is, at least for the start, no need for scaling features for the IDE component. The applications created and published using Arvue IDE are hosted on Amazon Web Services cloud to get high scalability.

As can be seen from the figure, Arvue offers all the tools needed for creating, publishing and hosting the applications. Furthermore, the application sources can be downloaded from the system and edited with an external IDE and then uploaded back to the repository.

Let us now go through the components one by one in the following subsections.

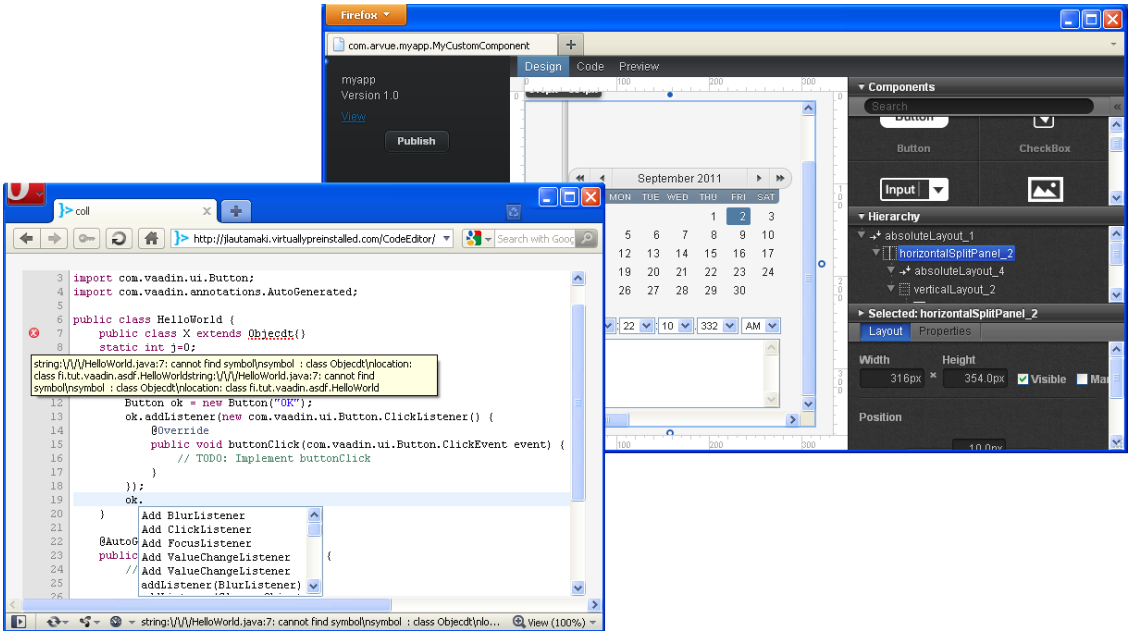

Figure 2. Screenshots of the Arvue IDE. Code editor at left and graphical UI designer at right.

4.1 Arvue IDE

Arvue IDE is a browser-based tool for creating Vaadin applications. Furthermore, the applications can be published to the cloud with a single click. Besides being used to create Vaadin applications, the IDE itself is a Vaadin application and could run within the same infrastructure as applications created with it. Arvue IDE contains a visual UI designer and a code editor CoRED (Collaborative Real-time Editor) [11], along with tools for previewing and publishing applications.

In the UI designer presented in Figure 2 (right), the user can create user interfaces by dragging and dropping Vaadin components such as layouts, containers and various kinds of UI elements to the middle panel. The position and size of the elements can be set using a simple drag and drop based interface. Further properties of an element, such as its style and whether it is enabled or read-only, can be set by selecting the element and defining its properties in a side panel view. In addition, there is also a hierarchical view of the component structure of the created UI.

Java source code for the designed application UI is automatically generated. The code can be further edited in the code editor illustrated in Figure 2 (left). With the editor, the user can add listeners for the UI components to attach functionalities. The UI components can also be added and modified in the code editor. The changes created to the code are reflected back to the UI designer, making a round trip between the UI designer and the code editor.

The code editor component offers various features to help the programmer. It uses Ace editor for Java syntax highlighting, automatic indentation and other basic code editor features. Additionally, our code editor checks errors in the user code. Once a while the code is compiled on the server side using Java Development Kit (JDK) compiler. The possible errors are presented in the code editor along with the error descriptions from the compiler. In addition the editor offers an automatic code completion. For example, when the user types "`myButton`.", a list of the fields and methods of the `myButton` object is shown for the user to select. In addition to the field and method suggestions, there are some Vaadin-specific suggestions. All the suggestions are also generated on the server side using JDK tools. Furthermore, CoRED can work in collaborative mode, meaning that several developers can edit the same source file simultaneously. For further details on the features and

implementation of the code editor, we refer to [11].

4.2 Version Control

The code generated by Arvue IDE is saved to Git[21] distributed revision control system. Git is meant for managing revisions in a file that has frequent changes possibly by multiple users. The system allows rolling back to previous revisions or assigning creator information for code lines of a file. In practice, Arvue version control system is implemented as an easy-to-use wrapper library built around the JGit[22] that handles all the operations necessary to interact with the repository.

One of the primary features of the version control component is automatic saving. It automatically commits code changes to Arvue Git repository and generates associated commit messages. Hence, the revisions can be tracked later on. The component also includes features like manual saving and loading of previous work from a specific external repository. This can be done with a simple user interface in the general view of the editor.

4.3 Publishing of Arvue Applications in the Cloud

The Arvue auto scaling framework takes care of sharing the workload on the distributed servers. It serves the web applications end user sessions to a scalable application server tier. The server level is dynamically scaled according to the workload in the typical cloud fashion. The framework consists of a number of subcomponents as shown in Figure 3.

An application server instance runs on a dynamically provisioned virtual machine (VM). Each application server runs multiple web applications under OSGi [17] environment. In OSGi environment, the web applications are run as specific components, called OSGi bundles. Bundles can be loaded and unloaded in dynamic fashion and they can communicate with each other.

There are multiple implementations for the OSGi standard [17]: e.g., free open-source Apache Felix[23] and Eclipse Equinox[24] as well as commercial ones like Makewave Knopflerfish[25]. From the open-source implementations, Apache Felix is certified to be compliant with the OSGi specification. Thus, we selected to use this one.

In addition to web applications, each application server also runs a local controller. The local controller logs application server and application performance. The main performance metrics are load average and resource consumption. The application specific data is generated by the Resource Manager which is described in 4.4. On regular intervals, each local controller sends performance data to the global controller as can be seen in Figure 3. Another task of the local controller is to control OSGi Felix for loading and unloading of web applications.

The compiled Arvue applications are stored in an application repository. When HTTP load balancer receives a new user session request for a web application not deployed on any application server, it directs the request to the global controller that selects a server and forwards the HTTP request. This causes the server to load the web application from the application repository. After a period of inactivity, the application is unloaded from the server.

The global controller acts as the capacity manager for application servers. Management includes starting and stopping application server instances when there is a need for scaling of the service.

[21] http://git-scm.com
[22] http://www.eclipse.org/jgit
[23] http://felix.apache.org
[24] http://www.eclipse.org/equinox
[25] http://www.knopflerfish.org

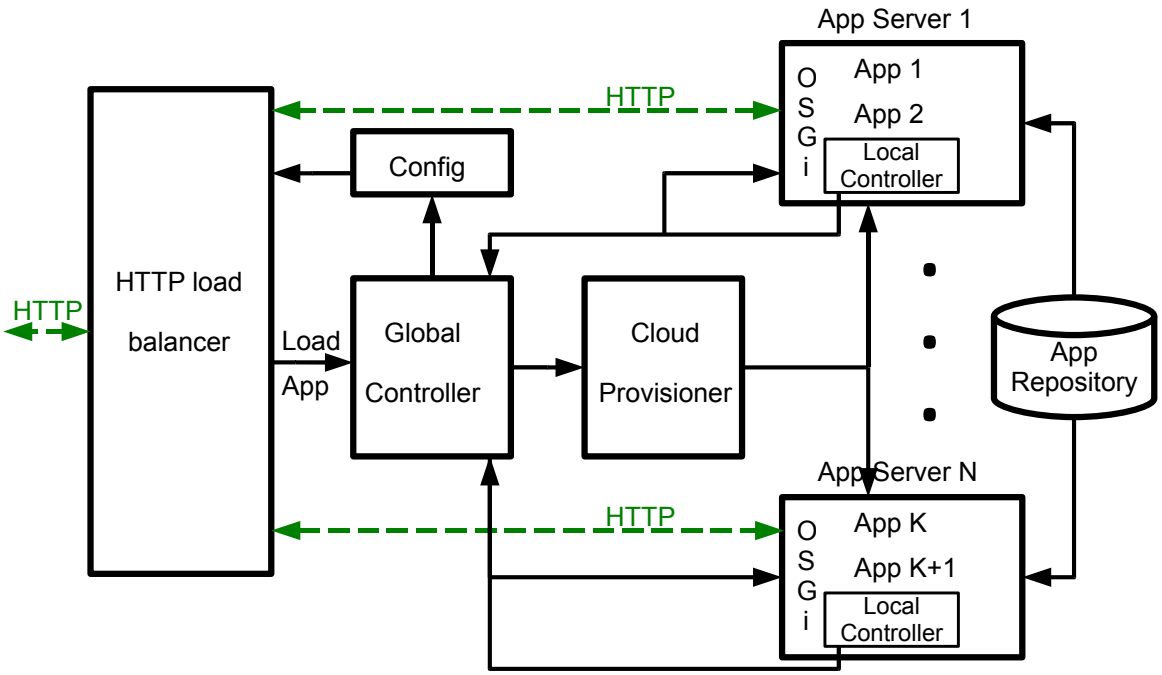

Figure 3. An abstract view of Arvue auto scaling framework.

While the scaling decisions are made by the global controller, the actual lower level tasks are done by a cloud provisioner. In our case, the provisioner is Amazon Web Services cloud, Amazon EC2[26].

All the HTTP requests are routed through a high performance HTTP load balancer and proxy. This way HTTP request load is balanced among the application server instances. As an implementation of the balancer, we use HAProxy[27]. For its functions, the balancer maintains configuration information about application servers that are running and about the web applications executed on each of them.

4.4 Security Issues on the Application Servers

Despite its many merits, OSGi unfortunately does not solve all the security problems attached to hosting multiple untrusted applications. In addition, the hosted applications are publicly available in the web which makes the situation even trickier. Some of the vulnerabilities are implied by Java principles [8, 2] and some by the OSGi features [14, 15]. The security issues can be splitted in two categories: the permission control of untrusted OSGi bundles and the more complicated matter of resource usage monitoring and controlling.

Because of the application environment, we are able to solve permission-based vulnerabilities moderately simply with the OSGI permissions. The OSGi Security model is based on the Java 2 Specification [16]. The basic idea is that code can be forced to authenticate before it has a

[26] http://aws.amazon.com/ec2
[27] http://haproxy.1wt.eu

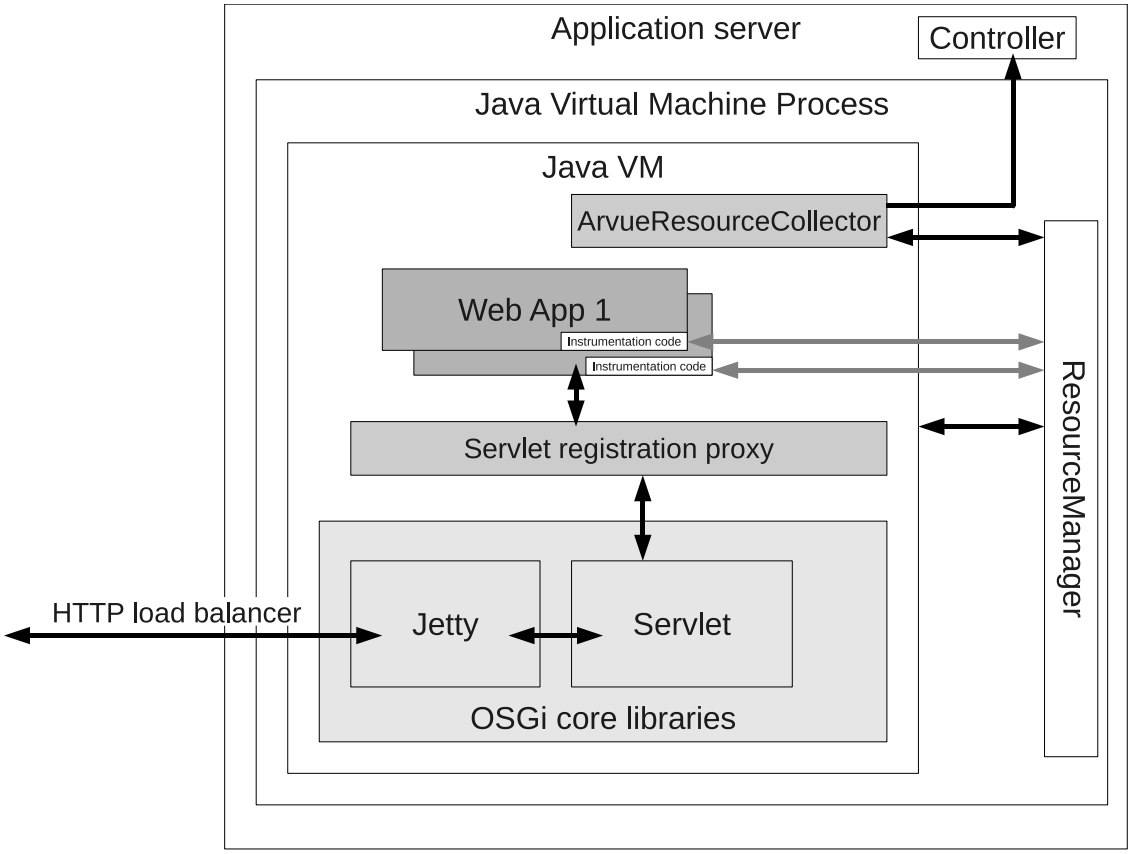

Figure 4. Resource Manager component

permission to execute specific method calls. In our case, authentication by bundle's location is enough. We refer to literature [6, Section 14.7] for more details on the permissions.

For monitoring resource consumption, we have implemented a monitoring component called Resource Manager. In practice, Resource Manager is a simple profiling tool, which is augmented with per-application limits such as memory allocation or CPU time used. When one of the limits is reached, Resource Manager tries to interrupt or stop the misbehaving application. In addition to monitoring, it provides resource usage information for the global controller via local controller.

The Resource Manager contains three different components as shown in Figure 4: the dynamic servlet registration bundle with a service call proxy to add monitoring aspect to web applications, Resource Manager for actual profiling tasks, and ArvueResourceCollector to capture the resource usage data from Resource Manager component.

Resource Manager is based on the Java Virtual Machine Tool Interface (JVMTI) [13] that provides a way to inspect the state of applications running in the Java virtual machine. The Resource Manager is hooked to the Java virtual machine so that when the virtual machine starts a new thread, the Resource Manager is called by the JVMTI. Similarly, it is called when threads are stopped. Memory allocations can be easily covered by instrumenting the code to call the Resource Manager via Java Native Interface (JNI) [12] whenever a new object is allocated. The instrumentation is carried out when the class file is loaded, so no preliminary changes to the code are required. To track the release of the memory JVMTI callbacks are used. CPU time is monitored with a separated thread that collects periodically (like once a second) the used CPU time on all the active threads.

5 Concluding Remarks

In this paper, we described a web based integrated development environment and hosting service called Arvue. With the introduced system it is fast and easy even for the beginner without any specific tools installed on the computer to create and publish Vaadin [4] based web applications.

The most visible part of Arvue is a browser-based IDE. The IDE contains a visual UI designer and a code editor along with an easy way to preview and publish applications. The applications are stored in an integrated version control system and can be published to cloud for anybody to access with a web browser. We also identified the challenges related to running user-created applications and gave our solutions for monitoring and controlling the resource usage of applications and for scaling the system up and down. Decisions concerning the security problems of user-created web applications are also illustrated.

There are plenty of browser-based development tools that have at least some of the features of Arvue. However, Arvue is intended for a neatly targeted use: creating and publishing small web applications for Arvue Java framework. The goal of Arvue is to take the process from beginning to publishing and hosting to the web. Thus, we are able to serve well suited tools for this exact need and make the process as flexible as possible. For example, we offer a graphical editor for creating the Vaadin application user interfaces. In addition, the Vaadin web applications can be developed purely with Java. There is no need for combining different languages and web development techniques. This is in contrast with the previously available tools which are meant for more general application development.

Since Arvue is still in the early proposal phase, we have numerous directions for future improvement. The collaborative capabilities of the tool have to be improved; in addition to those already implemented in the editor component as described in [11] we should have more tools for project management and for architectural design. Another obvious direction for future work is to perform a series of usability studies in order to find out how programmers wish to use the system. Based on the results, we can further refine the implementation and focus on the parts that provide most support for the actual development work.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. Communications of the ACM 53, 4 (2010), 50–58.
2. Geoffray, N., Thomas, G., Muller, G., Parrend, P., Frénot, S., and Folliot, B. I-JVM: A Java virtual machine for component isolation in OSGi. In Proceedings of the IEEE/IFIP International Conference on Dependable Systems & Networks (DSN) (2009), IEEE, pp. 544–553.
3. Goldman, M., Little, G., and Miller, R. C. Real-time collaborative coding in a web IDE. In Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST) (2011), ACM, pp. 155–164.
4. Grönroos, M. Book of Vaadin, 4th ed. Uniprint, Turku, Finland, 2011.
5. Hall, M. S. Java WIDE - Java Wiki Integrated Development environment: Nifty tools and assignments. Journal of Computing Sciences in Colleges 27, 1 (2011), 91.
6. Hall, R. S., Pauls, K., McCulloch, S., and Savage, D. OSGi in action: Creating modular applications in Java. Manning Publications, Greenwich, CT, 2010.
7. Hartmann, B., Dhillon, M., and Chan, M. K. HyperSource: Bridging the gap between source and code-related web sites. In Proceedings of the Annual Conference on Human Factors in Computing Systems (CHI) (2011), ACM, pp. 2207–2210.

8. Java Community Process. Java specification request 121: Application isolation API specification, 2006. Version 2.7, final.
9. Java Community Process. Java specification request 315: Java servlet 3.0 specification, 2009. Version 3.0, final.
10. Jenkins, J., Brannock, E., and Dekhane, S. JavaWIDE: Innovation in an online IDE (tutorial). Journal of Computing Sciences in Colleges 25, 5 (2010), 102–104.
11. Lautamäki, J., Nieminen, A., Koskinen, J., Aho, T., Mikkonen, T., and Englund, M. CoRED—Browser-based collaborative real-time editor for Java web applications. In Proceedings of the 15th ACM Conference on Computer Supported Cooperative Work (CSCW) (2012), ACM.
12. Oracle. Java native interface specification, 2006. Version 6.0.
13. Oracle. JVM tool interface, 2006. Version 1.2.1.
14. Parrend, P. Software Security Models for Service-Oriented Programming (SOP) Platforms. PhD thesis, Institut National des Sciences Appliquées de Lyon, France, 2008.
15. Parrend, P., and Frénot, S. Java components vulnerabilities: An experimental classification targeted at the OSGi platform. Tech. Rep. 6231, Institut National de Recherche en Informatique et en Automatique, Le Chesnay Cedex, France, 2007.
16. Sun Microsystems. Java 2 security architecture, 2002. Version 1.2.
17. The OSGi Alliance. OSGi service platform: Core specification, 2009. Release 4, version 4.2.
18. van Deursen, A., Mesbah, A., Cornelissen, B., Zaidman, A., Pinzger, M., and Guzzi, A. Adinda: A knowledgeable, browser-based IDE. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE) (2010), vol. 2, ACM, pp. 203–206.
19. Wu, L., Liang, G., Kui, S., and Wang, Q. CEclipse: An online IDE for programing in the cloud. In Proceedings of the IEEE World Congress on Services (SERVICES) (2011), pp. 45–52.

PUBLICATION VI

Janne Lautamäki, Antti Nieminen, Johannes Koskinen, Timo Aho, Tommi Mikkonen, and Marc Englund. "CoRED: browser-based Collaborative Real-time Editor for Java web applications." In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, pp. 1307-1316. ACM, 2012.

CoRED – Browser-based Collaborative Real-Time Editor for Java Web Applications

Janne Lautamäki, Antti Nieminen, Johannes Koskinen, Timo Aho, and Tommi Mikkonen
Tampere University of Technology
Korkeakoulunkatu 10, FI-33720, Tampere, Finland
{janne.lautamaki, antti.h.nieminen, johannes.koskinen, timo.aho, tommi.mikkonen}@tut.fi

Marc Englund
Vaadin Ltd.
Ruukinkatu 2-4, FI-20540, Turku, Finland
marc.englund@vaadin.com

ABSTRACT

While the users of completed applications are heavily moving from desktop to the web browser, the majority of developers are still working with desktop IDEs such as Eclipse or Visual Studio. In contrast to professional installable IDEs, current web-based code editors are simple text editors with extra features. They usually understand lexical syntax and can do highlighting and indenting, but lack many of the features seen in modern desktop editors. In this paper, we present CoRED, a browser-based collaborative real-time code editor for Java applications. CoRED is a complete Java editor with error checking and automatic code generation capabilities, extended with some features commonly associated with social media. As a proof of the concept, we have extended CoRED to support Java based Vaadin framework for web applications. Moreover, CoRED can be used either as a stand-alone version or as a component of any other software. It is already used as a part of browser based Arvue IDE.

Author Keywords
Development tools, collaboration architectures, Vaadin.

ACM Classification Keywords
D.2.3.c. Program editors. D.3.2.h. Development tools. H.5.3.c. Computer-supported cooperative work. J.8.s. Web site management/development tools.

General Terms
Design, Experimentation.

INTRODUCTION

It is widely recognized that communication problems are a major factor in the delay and failure of software projects [2]. Numerous tools and methods have been proposed to solve issues in different phases of projects, starting from capturing requirements and ending at customer documentation. One of the most promising approaches to communication problems is offered by agile methods that advocate close and frequent communication between the client and the developers. In reality, this is often implemented in the form of a team that shares the same premises, encouraging frequent informal communication.

While the software development community is already struggling with communication issues, the emerging practice of global software engineering is raising even more challenges: Software work is undertaken at geographically separated locations across national boundaries in a coordinated fashion, involving both real time (synchronous) and asynchronous interaction [13]. This emphasizes the need for timely, precise and uniform forms of communication across the planet. Then, since the development takes place at the global scale, also the necessary communication should take place at such scale.

In almost any other field, the recent standard answer to global communication problems has been the World Wide Web, or simply the Web. Indeed, in a relatively short time, the Web has become the platform for all types of applications that enables real-time collaboration in forms and scale that would have been difficult to imagine a few decades ago. Recently, the collaborative capabilities of the Web have been further enriched with a new invention – social media. Facebook, Linkedin and other services enable us to be in contact with the friends, colleagues, and enthusiasts of different topics in real time all over the planet.

While the users of completed applications are heavily moving from desktop to browser, the majority of developers are still working with desktop IDEs such as Eclipse or Visual Studio. At present, most of the available web-based code editors are just text editors with some extra features like code highlighting, indentations and collaboration clued on top and they are not yet as usable as the best of the desktop editors. However, the web editors offer their own possibilities. For example, real-time collaborative editing sits very naturally in the environment. In addition, we get the general web-based application

CSCW'12, February 11–15, 2012, Seattle, Washington, USA.

benefits like automatic distribution, installation and updating of applications [9] and independence on the development environment. Steps towards the direction are also proposed in [1, 14].

In this paper, we describe an experiment where the collaborative capabilities of the Web in general and the features of social media in particular are harnessed to help solving some of the communication problems of software development. As a concrete technical contribution, we introduce the browser based editor CoRED[1] (Collaborative Real-time Editor) intended for collaborative real-time editing of Java based source codes. CoRED contains highlighting, indentation, semantic error checking and some code completions. In addition, we present a number of features inspired by social media services, which we believe will be helpful for developing software applications. To the best of our knowledge, the similar set of functionalities is not offered by any of the previously existing web-based code editors.

We have selected Java and Vaadin [6] as the target language and framework, meaning that CoRED is an editor initially aimed for those two. Nevertheless, it can be extended for other environments with reasonable amount of work. The web applications based on Vaadin are implemented just like desktop Java applications. Because of the strong typing and good tool support in Java it is possible to augment CoRED with many features. These features include semantic error checking and code completion. This could just be dreamed of for weakly typed dynamic languages like JavaScript.

The rest of this paper is structured as follows. In the next section, we give an overview to the Vaadin framework, to the Ace editor, and to Java Developer Kit (JDK) which we are using as a base of our work. We also give a brief introduction to Arvue IDE which is going to use our editor as a building block. After this, we describe the technology behind our solutions and discuss the collaborative features. Moreover, we explain how CoRED could be extended for different frameworks. Then, we compare our editor with other code editors available in the web. Towards the end of the paper, we draw some final conclusions.

BACKGROUND

In this section, we give brief introduction to tools we are using in CoRED. We also show Arvue IDE as an example because it is going to be the first actual web application using CoRED. Tools we are using are the Vaadin framework, Ace editor, and Java Developer Kit (JDK). As a framework, Vaadin is obviously the lowest layer of our architecture. Vaadin is used for communicating over HTTP(S) and for making a separation of concerns between the client and the server [18]. On the client side, we use Ace editor as a front-end, and on the server side JDK as a tool for analyzing the source code. Vaadin also offers us a way for packaging the whole CoRED as a deliverable component as presented in Figure 1. CoRED can be used as a part of Vaadin based application like Arvue IDE or as it is.

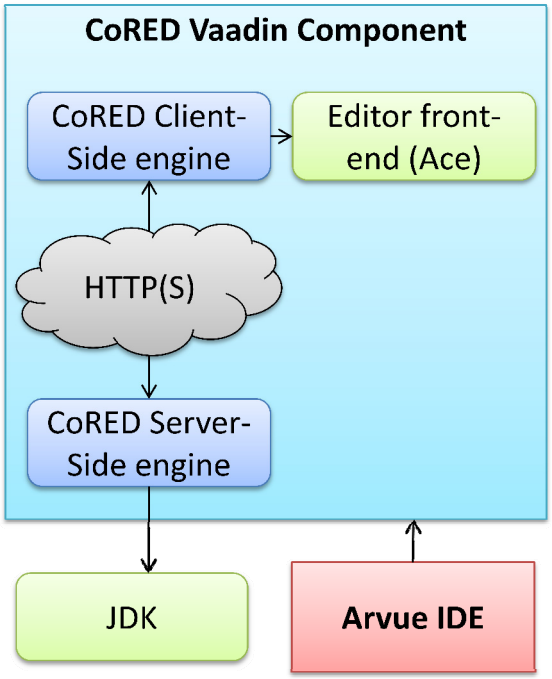

Figure 1. Architecture of CoRED Vaadin component and Arvue IDE using it.

Arvue

Arvue[2] is a cloud based IDE and hosting solution for the users who need a simple web-based tool for implementing and publishing Java based Vaadin applications. The basic philosophy of Arvue is to create applications "in the web for the web". In other words, the goal is to implement web applications in the web and publish them with minimal effort. No other tool except the browser is needed, implying that no installation is necessary in any phase of the development.

Applications are created in the browser-based visual editor that contains both the GUI (see Figure 2) and the code editor. On the GUI side, the user can create a new GUI just by dragging and dropping elements and layouts. The created GUI is then converted to source code and it can be further modified with the code editor (namely CoRED). Arvue is a round trip tool between the GUI and the text editor, meaning that same GUI edits can be done from both of the editors. The code editor is also used to implement features behind the GUI. Finally, the new application can effortlessly be deployed to a cloud type environment, which

[1]CoRED is available for testing at http://jlautamaki.virtuallypreinstalled.com/CoRED

[2]Arvue@dev.vaadin.com wiki. http://dev.vaadin.com/wiki/Arvue

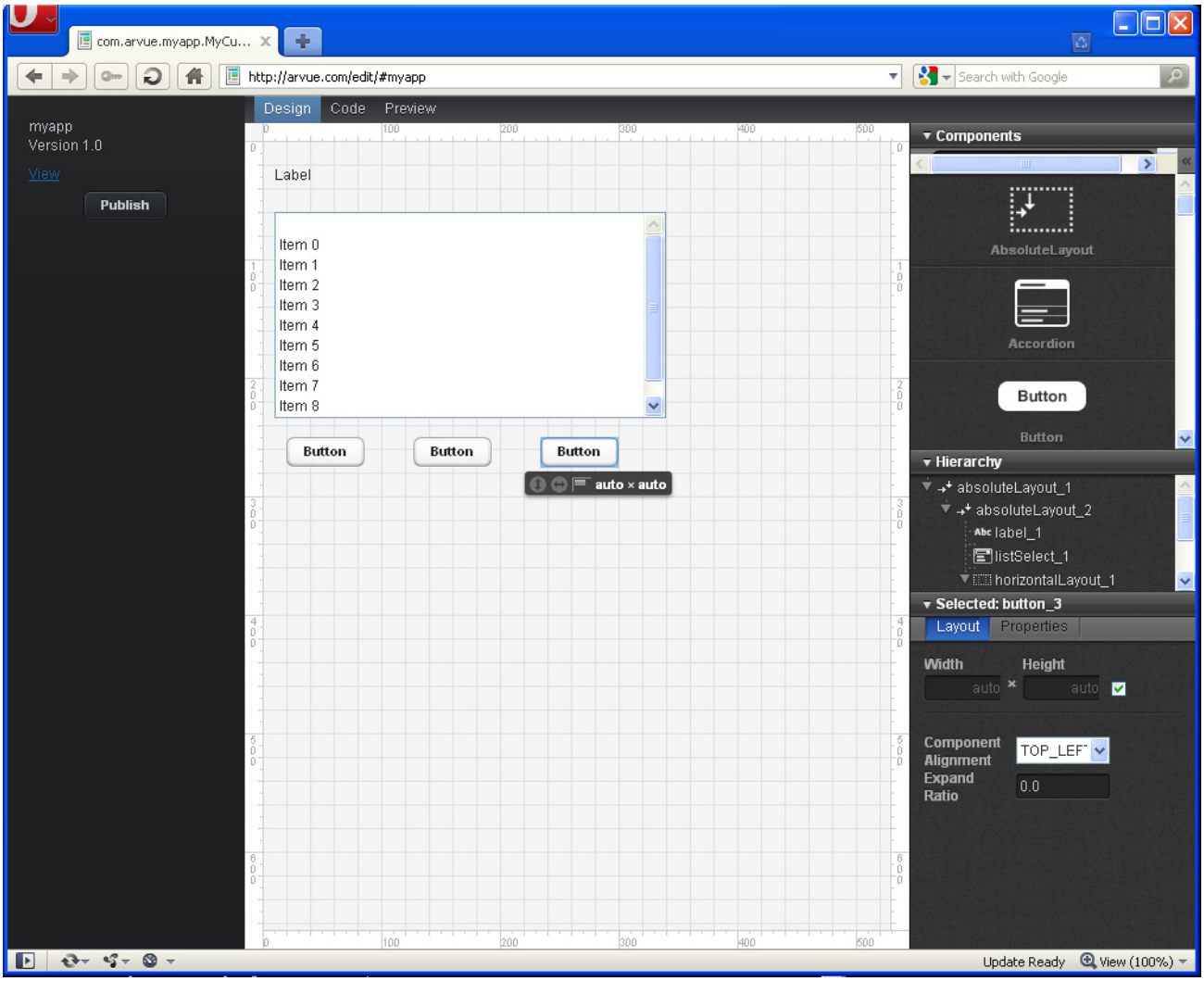

Figure 2. Arvue IDE with the graphical UI editor tab opened.

scales up by starting more server instances when needed. Every server is able to host multiple applications, which share many resources while not interfering with each other.

Arvue utilizes the Vaadin framework in its implementation. While editing is done inside a browser, most of the processing is carried out on the server side, as is common in Vaadin based system. For example, the server side of the CoRED utilizes JDK for semantic error checking and it also eases the implementation of code completion. Furthermore, our code editor is implemented as a custom Vaadin component whose client side is a Google Web Toolkit (GWT) [12] widget. Additionally, the client side widget uses JavaScript based Ace editor for basic editor capabilities.

The Vaadin Framework

Vaadin is an open source framework for developing Rich Internet Applications (RIA) using the Java programming language. The Vaadin framework relies extensively on the facilities of GWT [12]. GWT is an open source development system that allows the developer to write AJAX applications [10] in Java and then compile the source code to JavaScript which can be run on all browsers. In the Vaadin framework, GWT is used for compiling the client side engine and for communication between the client and the server.

From the developer perspective, individual Vaadin applications are implemented similarly to Java Standard Edition desktop applications. However, instead of usual UI libraries like AWT, Swing or SWT, the developer has to use the specific set of Vaadin UI components and the framework knows how to use the browser as a view. In addition, new custom made UI components can be implemented. In this case the client side of the customized UI component can be either developed in Java and then compiled with GWT or written directly with JavaScript. The use of any combination of Java and JavaScript is also possible. Thus, this enables us to use readily made JavaScript applications as a part of the Vaadin client side component.

Java Developer Kit (JDK)

Usually, Java applications run on top of a Java Runtime Environment (JRE) and are only compiled using a Java bytecode compiler included in developer kits like the Sun JDK[3]. However, this is not enough in order to develop the code editor applications, which can create other new applications on the fly. Instead, we need to run the application on the developer kit. JDK contains the necessary tools for compiling, executing, debugging and documenting the programs. Furthermore, it includes `tools.jar` package, which contains useful Sun specific APIs for compiling, diagnosing, and parsing the source code.

For our purposes, JDK provides a couple of very useful features: JavaCompiler[4] is an interface for invoking Java compiler from the program. The compiler generates complete error and warning diagnostics during compilation (for example, error messages) and they can be collected using DiagnosticCollector. In addition, by extending ClassLoader and StandardFileManager, the source and destination of compilation can be redirected. Finally, the offered API contains TreePathScanner[5] that can be used for processing source code. The scanner visits all the child tree nodes of the source code and can, thus, be used for finding Classes, Methods, Variables and other kinds of Java structures.

Ace

Ace[6] is an open source code editor written using JavaScript. It can be easily embedded in any web page and it has support for several different programming languages, including Java. Ace offers a lot of basic text editor features such as undo/redo, search/replace, and customization of appearance using themes. It also implements many features important specifically for programmers like syntax highlighting and automatic indentation.

It is easy to extend the behavior of Ace without editing its source code. It is possible, for example, to implement your own keyboard handler. Another feature important to us is that Ace supports markers for showing code errors. Also custom markers like underlining are possible. Moreover, Ace offers information useful in integrating the editor into a broader framework. For example, the position of the cursor in screen coordinates is needed for displaying a suggestion box at a suitable position.

ARCHITECTURE OF CORED

In CoRED, most of the hard work such as checking code errors and generating code suggestions is done on the server side. The client side editor does the interaction with the user. CoRED utilizes the Vaadin framework to tie these two sides together.

Separation of concerns

Both the client side and the server side of CoRED are designed to be easily customizable. Most of the features of the editor, such as error checking or code suggestions are implemented as replaceable and extendable components. A part of the CoRED architecture is presented in Figure 3. The main component, CollaborativeCodeEditor, and its client side counterpart act as glue between the server side components and the front-end editor. For example, when the user needs code suggestions, the main CoRED component requests suggestions from the server and then displays a widget for the user for selecting among the suggestions. The suggestions are generated by server side suggester components, and the selected suggestion is finally sent to the front-end editor.

The front-end editor component is typically a wrapper for a third-party JavaScript code editor. We have implemented a prototype component with three possible choices for the front-end editor: Ace, CodeMirror[7] and Eclipse Orion[8]. Wrapping a JavaScript editor inside a Vaadin GWT component was quite straightforward using JavaScript Native Interface (JSNI)[9] calls. Eventually, we chose Ace as our front-end editor because of its good support for indentation, syntax highlighting and customizable markers among other things.

CoRED has a possibility for flexible component add-ons. For example error checking and code completion components can be added by simply implementing the corresponding interfaces. Next we present the implemented components in detail.

Error Checking

Error checker is an example of component for extending CoRED. For checking errors, there are basically two possible approaches. First, we may use our own or third party library for parsing and error searching. Second, it is possible to compile the code from source to bytecode and then get compiling diagnostics.

For our implementation, we decided to use the latter solution with Java SE Development Kit (JDK). This way

[3]JDK File Structure for Windows. http://download.oracle.com/javase/6/docs/technotes/tools/windows/jdkfiles.html

[4]Java Compiler (Java Platform SE6). http://download.oracle.com/javase/6/docs/api/javax/tools/JavaCompiler.html

[5]TreePathScanner (Compiler Tree API). http://download.oracle.com/javase/6/docs/jdk/api/javac/tree/com/sun/source/util/TreePathScanner.html

[6]Ace – Ajax.org Cloud9 Editor. http://ace.ajax.org/

[7]CodeMirror. http://codemirror.net/

[8]Orion – Eclipsepedia. http://wiki.eclipse.org/Orion

[9]Coding Basics - JavaScript Native Interface (JSNI). http://code.google.com/webtoolkit/doc/latest/DevGuideCodingBasicsJSNI.html

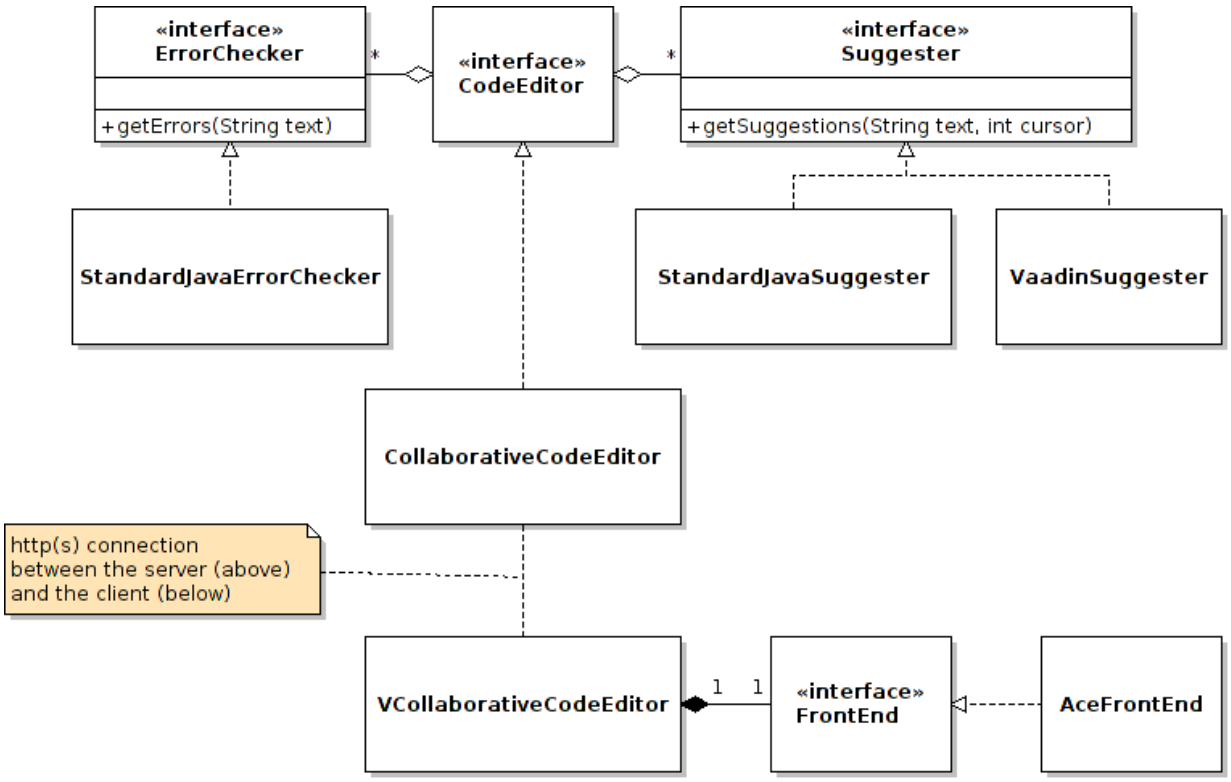

Figure 3. A part of the CoRED class hierarchy.

we get all-inclusive error and warning diagnostics from JavaCompiler and we do not need to stumble with the parser-compiler incompatibility. However, a basic problem with compiling is its consumption of computing resources. If we use a third party library or some kind of separate incremental parser like in Eclipse SDK for error checking, we could have better changes to tune the balance between efficiency and accuracy. However, we think that the approach is sufficient for our prototype.

In error checking, our basic procedure is to first compile the code and then ask for compiling diagnostics. Error diagnostics offered by JDK contain all the information needed for the message to user: in addition to the actual error message, line and column numbers are available. For efficiency reasons we do not save the files but only compile in memory. Furthermore, we compile only when we assume that the user wants the errors to be checked. In practice we wait for a while after the last edit and compile when user is in an idle state planning the next modifications. No compilation is done during code editing and even during the compilation the user interface is not blocked.

With just a few users, the compiling happens in the blink of an eye and most of the delay is caused by network communication. However, with multiple users, the continuous compiling is an efficiency problem. To make the system more scalable we compile less frequently when the number of users increases. In practice this is implemented with a separate worker thread for compilations.

Suggestions and Code Completions

While editing the source code, the code editor also suggests possible code completions. The suggestions can be invoked in two ways, by using a special key combination or by typing a dot after, e.g., an interface or an object name.

In either case, the server side client analyzes the code with all the implemented suggestion components. In fact, we have implemented two different suggestion components: one for standard Java suggestions and another for Vaadin specific ones. Based on the cursor location in the text the components resolve suitable suggestions. In addition to the actual inserted text, each suggestion includes a visible name of the suggestion and a longer description. These are passed to the editor and then shown to the user as seen in Figure 4.

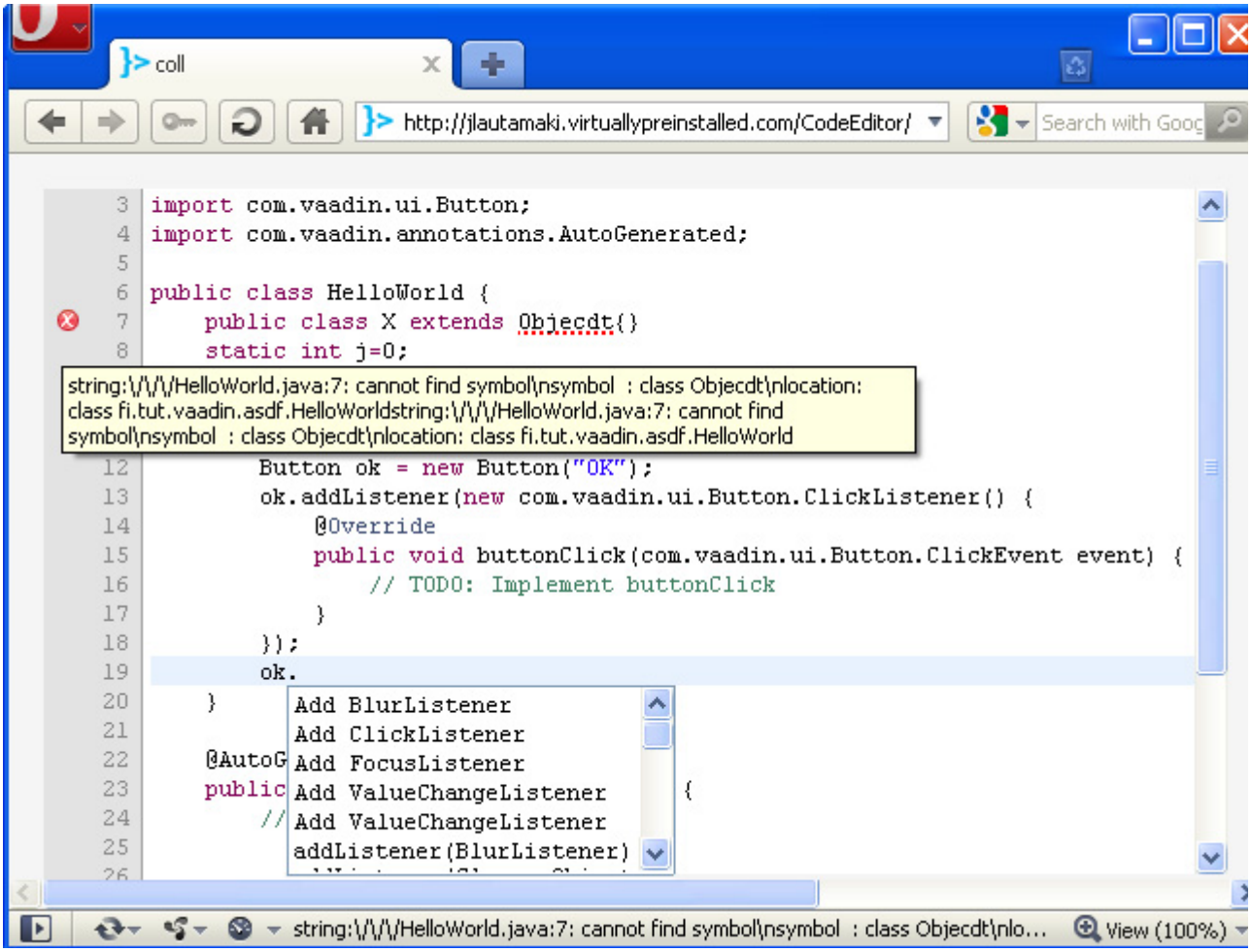

Figure 4. CoRED with generated listener, error tooltip and suggestion box opened (last two cannot normally be opened simultaneously)

To help resolving the suggestions, we use tools offered by JDK. With TreePathScanner it is possible to build a tree of classes, methods, variables or other parts of Java syntax. The currently visible variables, their types and methods, and the visibility scopes can be tracked down. To calculate the suggestions, the whole document is scanned with linear time operation. A suggestion may be related to a class originating from an imported package. For resolving those suggestions, we load the class and use reflection[10] to ask for its public methods and variables.

Collaborative features

It is natural to consider web applications as multiuser applications. The main requirement for a multiuser application is the shared data [11]. When all the users are connected to the same system it seems only reasonable to assume that in addition to interacting with the system, they are communicating with each other. In our case collaboration means simultaneous editing of code document and some features inspired by social media.

Collaborative editing

For collaborative editing, we decided to use Neil Frasers

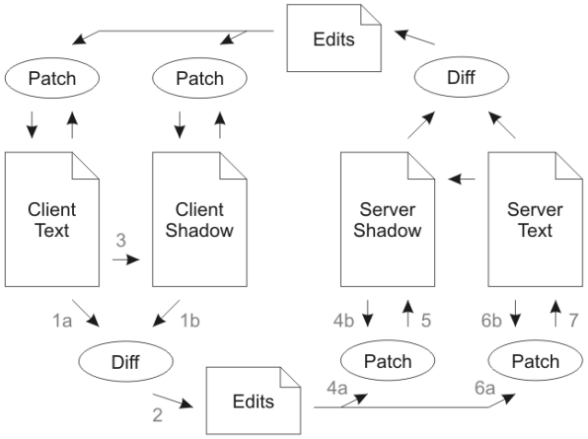

Figure 5. Neil Fraser's Differential synchronization with shadows [11]

[10]Reflection (Java SE Documentation) http://download.oracle.com/javase/6/docs/technotes/guides/reflection

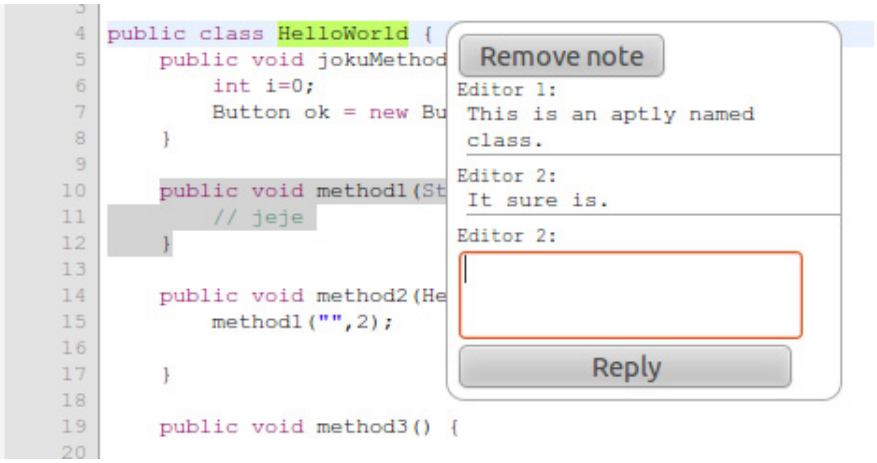

Figure 6 . A note related to "HelloWorld" and a reply in CoRED. The method called "method1" is locked by another editor.

Differential Synchronization with shadows [5]. It is a robust and convergent collaborative editing algorithm with open source implementations available on various languages, including Java and JavaScript.

The basic idea of Differential Synchronization is the following: the server stores the shared document, and each client has a separate shadow copy of the document both on server and client side, along with the copy they are editing (Figure 5). When a client changes its document, the differences between the newly edited document and the latest shadow are calculated using Myer's algorithm [8]. A patch (made using Bitap matching algorithm [18]) containing the differences is sent to the server. The patch is used both for keeping the server side shadow in sync, and for applying the changes to the shared document. The algorithm is symmetrical in a sense that the changes made to the shared document by other clients, is communicated to the client in the same way as the changes from the client to the server.

Differential synchronization is relatively easy to implement and it meets three basic demands often set for collaborative editing [16]: Firstly, it has high responsiveness. The edits can be done locally and only the differences are delivered to the server. Therefore the local actions are as quick as in a single-user editor. Secondly, it has high concurrency rate and multiple users can simultaneously edit any part of the document. Finally, it is able to hide communication latencies to some extent. Naturally, when latencies grow the conflict rate rises.

In addition to differential synchronization, there are plenty of ways to implement collaborative editing. One approach would be to lock the document or a subsection of the document before editing. A drawback with the locking approach is that a user must wait for the lock to be confirmed by a server, resulting to a loss in usability [15].

Another commonly used technique in collaborative editing is Operational Transform (OT) [3, 4, 7], used in e.g. Ethercodes[11] and Google Docs[12]. In OT, each *operation* (insert, delete, etc.) is recorded and sent to other clients who then transform the operation to take into account all the concurrently executed operations. In practical use, the OT implementations must deal with quite a large number of different editor actions including cut, paste, auto-corrections, auto-indentation, and suggestions, which may make the implementation problematic [5].

Features inspired by social media

In addition to plain collaboration in the form of writing code collaboratively, there are also some features inspired by social media. Perhaps the most obvious feature is the option to write comments in a fashion normally associated with text rather than code, as demonstrated in Figure 6. When some text is selected ("HelloWorld" in the figure), a popup is shown that allows the user to add a note referring to that text. When a user places the cursor on top of an existing note, the note popup is shown. Other users can reply to the original note, thus extending the note to a discussion, as also shown in Figure 6.

The notes reside in their original logical positions even though the document is modified. The start and the end positions of a note are marked with a *marker*. When a user inserts (or deletes) text before a marker, it is moved forward (or backward). The positions of markers are kept in synchronization similarly to the differential synchronization algorithm by sending, along with the text differences, also the position changes of markers from the client to the server and back.

[11] EtherCodes: Online Collaborative Code Editing. http://gigaom.com/collaboration/ethercodes-online-collaborative-code-editing/

[12] Google Docs. http://docs.google.com/

In CoRED, it is also possible to lock portions of the document to be edited only by one editor. For example, if a user wanted to make changes to a specific function without anybody interfering, he/she could lock a function by selecting the function and clicking a "Lock for me" button in a popup that showed up after selecting. After locking, the locked area is shown as grey for other users and they are unable to edit it although they can still add notes to it. Like notes, locks also have a start marker and an end marker which retain their logical positions.

The locking requests are sent to the server, and if no one else had an overlapping lock, the lock is granted. Users are disallowed to do any operations that would modify (or delete) a portion locked by another user. Such actions are already blocked on the client side, but the validity of edits is also checked on the server side. The additional server side check is needed because a lock may have been granted to editor A while editor B was editing the same part of the document. Thus, when B sends its modifications to the server, they are discarded.

Performance considerations

The most time-consuming operation on the server side CoRED is the compiling of the edited Java source code. The result of the compilation is needed to present the user with error messages and code suggestions.

In our brief experiments, when running the server on a laptop containing Core 2 Duo processor, the compilation of a typical Java file with a couple of hundreds of lines, took tens of milliseconds. For sufficient user experience, the compilation needs to be done at most once a second, which could be managed on quite modest hardware.

Another heavy operation is applying patches to the collaborative document. The resource consumption of patching depends on the density of edits done on the document. It should be noted that there is a practical usability limit on how many people can edit the same document simultaneously, which is most likely a more constraining factor than the performance of patching. In our brief tests where a few people edited the same document simultaneously, there were no performance issues.

Editing multiple documents can be done on separate instances of CoRED, which could easily be distributed on multiple server machines if needed. Consequently we do not presently consider this as a major performance bottleneck.

CUSTOMIZING THE EDITOR

CoRED can be customized and extended in several ways. As mentioned, it is possible to use new error checkers and suggesters instead of, or in addition to, the existing components. For example, it is possible to use CoRED for another language than Java by implementing custom suggesters and error checkers, and changing the syntax highlighting and indentation of the front-end editor.

Suggestions

In case of Java based frameworks, it is possible to use our Java suggester. The required jar files of the target framework just have to be added to classpath. If the need for special suggestions related to a specific framework arises, it is possible to create custom suggesters by implementing the Suggester interface in Figure 3.

For example, when developing for Vaadin framework, there is often a need to add an anonymous listener for an UI component, such as ClickListener for a button. We have developed a Vaadin specific suggester, which is to be used in addition to the standard Java suggester. It generates empty skeletons for anonymous listeners where applicable. For example, let us assume that there is a Vaadin Button called myButton defined in the scope of the cursor. When the user types “myButton.”, one of the suggestions is to add an anonymous ClickListener. An illustration of the above case can be seen in Figure 4. The anonymous skeletons are created using Java reflection. Similar suggesters can be easily developed for other Java frameworks.

The suggestion feature is not limited to Java language, although we have not implemented any non-Java suggesters. It is possible to develop arbitrary suggesters for any language. However, creating a useful suggester for a dynamically typed language such as JavaScript is more difficult.

Error checking

As our error checker relies on the Java compiler, it can be used with any Java-based framework as long as the correct libraries are in the classpath. The standard Java error checking is most likely sufficient for most Java projects. Although it is possible to create custom error checkers by implementing the ErrorChecker interface (Figure 3).

Once again, things get more difficult with non-Java frameworks. Error checkers can be created for any language but CoRED offers no support for non-Java ones. The Ace editor we use as a front-end, contains a JavaScript error checker that is not a part of the CoRED architecture.

Front-end editor

Ace, the front-end editor used in CoRED, is highly customizable. It offers the possibility to change its appearance and define custom highlighting rules in separate configuration files. Ace already contains the files for the most popular languages such as JavaScript, HTML, XML, PHP, C++. Thus, adapting the front-end to be used for developing for other languages than Java is very simple.

If Ace, for some reason, does not meet our requirements, it is even possible to use another front-end, leaving the rest of CoRED intact. The front-end component must implement the FrontEnd interface (Figure 3). The interface defines the methods needed for the editor, including setting and getting the editor text, changing the cursor position, setting callbacks for changes, displaying error markers, and so on.

RELATED WORK AND COMPARISON

CoRED is not the first in the field of collaborative browser-based code editors. The first collaborative editor was presented as early as 1968 in the demo, retrospectively named as "The Mother of All Demos"[13]. Naturally, the demo in 1968 did not work on the browser. One of the first collaborative editors runnable in a browser (using Java Applets) was REDUCE [17]. The Web 2.0 phenomenon introduced the browser-based collaborative editors to a larger audience. One of the most successful was Writely that later evolved to Google Docs. The step from a text editor to simple code editor is quite short, and currently lots of browser based collaborative code editors are available.

Most of the browser based code editors have some of the same features as CoRED, but none has exactly the same ones. Many of the competing editors like CodeMirror and Ace are aimed for only code editing, not for creating applications. Like in the desktop world, also in the web there are wide variety of different languages and frameworks. Thus, it is virtually impossible to create one IDE that would support the whole project from writing the code from scratch to publishing it for all of the frameworks. As CoRED is a part of Arvue IDE, our goal is exactly the whole process from beginning to deploying to the web. Akshell[14] has the similar kind of framework approach. It allows a developer to implement both the client and the server with JavaScript. The system also takes care of deploying. In addition, Orion project by Eclipse community limits the frameworks that the user can use. Orion is still in a proposal phase but it looks promising.

When comparing the features of other editors, most code editors have code coloring and indenting available for several different programming languages. To the best of our knowledge, there is only one other example of the browser based editor with code completion and error checking. The editor, named WWWorkspace[15], uses Eclipse as a back-end. As our editor has probably some scalability issues because of JDK, one can only imagine the situation with a massive editor like Eclipse. Like CoRED, WWWorkspace is also using strongly typed Java language. Most of other code editor examples are aimed for weakly typed dynamic languages like JavaScript. JavaScript has its own good sides, but it is virtually impossible to do semantic checking for weakly typed interpretive language.

As a further comparison, CoRED can be easily extended to support any Java based framework. As a downside, CoRED is heavily dependent of the server side and it cannot be used in offline mode like, for example, Google Docs.

[13] Wikipedia – The Mother of All Demos. http://en.wikipedia.org/wiki/The_Mother_of_All_Demos

[14] Akshell. http://www.akshell.com/

[15] WWWorkspace. http://www.willryan.co.uk/WWWorkspace/

FUTURE WORK

Since we are still working with the editor, it is obvious that numerous directions to future work exist. To begin with, introducing support for other phases of development work, such as requirements or project management, would be natural extensions. Then, also these tasks could be turned collaborative as well as more closely linked with development activities. Currently, we assume that these features come from environments that use CoRED as a component.

Another obvious direction for future work is to perform the series of usability studies in order to find out how programmers wish to use the facilities of the system. This could take place in the form of a coding camp, where students would take part in the experiment, and provide feedback on the system. Based on the results, we can then further refine the implementation and focus on parts that provide most support for the actual development work. In addition, the results of such studies could also help us to identify more potentially useful features that are commonplace in social media but not widely applied in software development.

Finally, in order to gain experiences from a larger user base, we are going to publish CoRED in the open Vaadin Directory[16] as a re-usable add-on for all the Vaadin Community. CoRED can be used as a component of other Vaadin projects or as a stand-alone application. It will also be used as a part of the IDE called Arvue. Currently, Arvue is in an early alpha stage. The other components are going to be the graphical designer for generating user interfaces and the capability to save applications to Git version management. In addition the developed applications can be published directly to the offered cloud. Arvue is mostly designed for creating small Vaadin applications and for testing purposes, but there is no obvious reason why it could not be used more generally.

CONCLUSION

In this paper, we described the browser based editor CoRED intended for collaborative real-time editing of the Java based source codes. We extended the system to offer some Vaadin framework specific features for developing the web applications in the web. As editing features, CoRED contains highlighting, indentation, semantic error checking and code completion. As a combination, this set of features is unique when compared with other browser based code editors. CoRED is going to be published in the Vaadin Directory as a re-usable add-on. It will also be used as a part of the web based IDE called Arvue.

CoRED is built to be modular and many of its parts can be replaced or extended. In this paper, we gave a small summary of what kind of modifications are needed to extend CoRED to different frameworks. In the case of Java

[16] Directory – vaadin.com. http://vaadin.com/directory

based frameworks, just small additions for the suggestion component and error checker are needed. However, for non-Java frameworks more laborious modifications are needed and the benefit of using CoRED is smaller. Fortunately, communication from client to server and collaborative features should work without problems in frameworks of all kind.

As a further contribution, we discussed on how the web applications should be developed and made a small comparison between our editor and other web based code editors available. As a piece of future work it would be interesting to do more experimentation on the usability as well as the scalability of our system.

REFERENCES

1. Begel, A., DeLine, R., and Zimmermann, T. Social media for software engineering. In *Proceedings of the Workshop on Future of Software Engineering Research*, pp. 33–38. Santa Fe, NM, USA, 2010.
2. Curtis, B., Krasner, H., and Iscoe, N. A Field Study of the Software Design Process for Large Systems. Communications of the ACM 31(11), pp. 1268–1287. 1988.
3. Davis, A. H., Sun, C., and Lu, J. Generalizing operational transformation to the standard general markup language. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pp. 58–67. New York, NY, USA, 2002.
4. Ellis, C. A., and Gibbs, S. J. Concurrency control in groupware systems. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pp. 399–407. Portland, OR, USA, 1989.
5. Fraser, N. Differential Synchronization. In *Proceedings of the 2009 ACM Symposium on Document Engineering*, pp. 13–20. New York, NY, USA, 2009.
6. Grönroos, M. Book of Vaadin. Uniprint. 2011.
7. Ignat, C-L., and Norrie M. C. Customizable collaborative editor relying on treeOPT algorithm. In *Proceedings of the eighth conference on European Conference on Computer Supported Cooperative Work*, pp. 315–334. Norwell, MA, USA, 2003.
8. Myers, E. W. An O(ND) difference algorithm and its variations. Algorithmica 1(1), pp. 251–266. 1986.
9. O'Reilly, T. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. O'Reilly. 2005.
10. Paulson, L. D. Building rich web applications with Ajax. Computer 38(10), pp. 14–17. 2005.
11. Patterson, J. F., Hill, R. D., Rohall, S. L., and Meeks S. W. Rendezvous: an architecture for synchronous multi-user applications. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pp. 317–328. New York, NY, USA, 1990.
12. Perry, Bruce W. Google Web Toolkit for Ajax. O'Reilly Short Cuts. O'Reilly, 2007.
13. Sahay, S. Global Software Alliances: The Challenge of 'Standardization'. Scandinavian Journal of Information Systems 15, pp. 3–21. 2003.
14. Storey, M-A., Treude, C., van Deursen, A, and Cheng, L-T. The impact of social media on software engineering practices and tools. In *Proceedings of the Workshop on Future of Software Engineering Research*, pp. 359–364. Santa Fe, NM, USA, 2010.
15. Sun. C. Optional and Responsive Fine-Grain Locking in Internet-Based Collaborative Systems. IEEE Transactions on Parallel and Distributed Systems 13(9), pp. 994–1008. 2002.
16. Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. Achieving Convergence, Causality-Preservation, and Intention-Preservation in Real-Time Cooperative Editing Systems. ACM Transactions on Computer-Human Interaction 5(1), pp. 63–108. 1998.
17. Sun, C., Jia, X., Zhang, Y., Yang, Y., and Zhang, Y. REDUCE: a prototypical cooperative editing system. In *Proceedings of the Seventh International Conference on Human-Computer Interaction*, pp. 89–92. 1997.
18. Sun, W. Manbed, U. Fast text searching with errors. Communications of the ACM, 35(10), pp. 83–91. 1992.

PUBLICATION VII

Juha-Matti Vanhatupa and Janne Lautamäki. “Content Generation in a Collaborative Browser-Based Game Environment.” Handbook of Digital Games, ISBN: 978-1-118-32803-3, p. 26, Wiley-IEEE Press, 2014.

[Book Title], Edited by [Editor's Name].
ISBN 0-471-XXXXX-X

Chapter 0

Content Generation in a Collaborative Browser-Based Game Environment

Juha-Matti Vanhatupa

Janne Lautamäki

Department of Software Systems
Tampere University of Technology

Abstract

Procedural content generation is a common method in installable computer games. It has been used from early rogue-like computer role-playing to modern strategy games for generating dungeons and world maps. However, procedural content generation is not well explored in browser environment. This chapter presents an approach using content generation methods to create content for a multiplayer browser-based game. The proposed approach has been applied to build a collaborative browser-based fantasy themed game, where players join their forces to complete quests generated by the game. All the quests in the game are generated dynamically at run-time based on simple quest templates. Our example game is implemented using client and server-side JavaScript. Presented methods can be used as a supplement to pre-created content. The generated content can be used for expanding overall game content and for increasing the replayability of the game.

1. Introduction

Currently, the software industry is experiencing a paradigm shift from conventional (binary) software towards web-based software. More and more applications are written for the Web and universally accessed by any device with a Web browser. Ideally, these applications will also support user collaboration by allowing multiple users to interact by using the same application and shared data over the Web [1]. Computer games are also experiencing the same paradigm shift. In recent years the quality of browser-based games has increased, bringing those closer to traditional computer games.

Today, browser-based games are an emerging genre, and browser-based environments are one of the upcoming trends of gaming. Browser-based games need no binary installations; therefore they are available for the huge number of potential players that necessarily have not ever owned a single computer. They are also available for any computer or mobile phone connected to the Internet. Furthermore, updating is an easy task; a developer can update a running application on a server and modifications spread out immediately [2].

Currently there is a myriad of technologies that can be used to implement browser-based games [3], as well as for web application development in general. In addition many web applications and games are developed using a combination of several technologies, which makes the application development even more complex. Partly this complexity derives from the fact that the Web is strongly rooted in information sharing and the original technology stack does not offer much support for a much more dynamic approach that web applications represent. However, as web development technologies evolve and become more sophisticated, the most likely the number of different technologies needed for developing a single web application will decrease. Some attempts toward using single language for the whole web application have been taken. For example, in GWT [4] and Vaadin [5] approaches, the whole application is implemented using just Java, and with NodeJS [6], both the client and server are implemented with JavaScript.

Browser-based games containing a persistent game world gather huge online communities since the same players are playing the game for a long time [7]. The players are using forums, wikis, social media, and IRC-channels to connect each other. An active

online community can be very strong motivation for a player to play a certain browser-based game. Although online communities mostly form along of long-term browser-based games, short-term browser-based games can also gather a faithful player group. Instead of strict clans or teams, these players form looser communities that constitute a huge mass of potential players for all browser-based games. Recent popularity of social networking services like Facebook, which are also application platforms where browser-based game can be deployed, helps to gather potential players [8]. The application platform can automate the registration process. Furthermore, it offers tools for users to advertise games by sending invitations through the platform or simply by telling others which games they liked.

"The honeymoon period" is a common phenomenon in browser-based games [9]. When a player starts to play a new game, the beginning is usually exiting and new. There are many things to learn and new places to explore. The player learns how the game is played, and depending on the game type, starts to gather experience for his character or to develop his empire. The honeymoon period can last for weeks or even months, but as the player's character levels up or the empire of expends, the initial fascination can quickly fade unless new challenges appear. We claim that procedural content generation can be used to expand the length of the period by creating new challenges and later ease the disappointment of the end of the honeymoon period.

The sizes of computer games have been constantly growing. Consequently, the sizes of development teams have grown and the time needed to develop of a single computer game has multiplied. Production of game content can be seen as one of the main challenges in the project. Nowadays a typical time span from an idea to a finished product is about two years and demands the work contribution of 20-50 people [10]. Massive amount of this time and resources are spent by programming the game content.

Content generation can be used to alleviate the burden that software developers experience in manual content creation. *Procedural content generation* refers to content generated algorithmically instead of manual creation. It is a common method in installable computer games and has been used from generating dungeons in early rogue-like computer role-playing games to world map generation in modern strategy games. Compared with desktop based games, the most of browser-based games still contain quite

a limited amount of content and maybe therefore the content generation is not well explored among them. However, there are no technical limitations and as the browser based games are becoming more complex, we speculate that content generation is becoming relevant quite a soon.

This chapter looks at possibilities and limitations of procedural content generation in browser-based games. We present a prototype computer role-playing game (CRPG) relying heavily on content generation. In the game, fantasy characters like warriors and wizards are used for accomplishing quests generated by the game. The quests consist of different types of rooms filled with fantasy monsters and treasures. All the quests in the game are produced dynamically at run-time based on simple quest templates. Furthermore, the creation of new quest templates is designed to be simple and straightforward. The main contributions of this article are: an approach using procedural content generation at run-time in browser-based game environment, a realization of this approach and lessons learned from making this implementation. We aim to prove that run-time procedural content generation, JavaScript scripting language, NodeJS framework, and used software libraries are very applicable tools for making browser-based games.

The implementation language of our prototype game is JavaScript. NodeJS framework allows using JavaScript at both client and server-side, and NowJS [11] library helps us to implement multi-player aspects into the game. The game flow is turn-based, however in this multiplayer environment a single player has only a few seconds to carry out his turn, after which all actions are processed and revealed to the other players at the same time. Most player actions are resolved at the server; however, a database is used as a permanent storage for collected experience points and the treasures. The players work together to complete the quests produced by the game. Although the game is a simple prototype, it demonstrates well the outstanding possibilities of content generation in a browser environment.

The rest of this chapter is structured as follows. In the next part of this chapter, we give and an abstract overview on how the Web can be utilized as a gaming platform. After that, the approach is extended for the multi-player games. In fourth part we describe content generation and explain how it can be used for both, reducing work needed for

implementing the game and for increasing the replayability of the game. In fifth part of this chapter, our example game called Caves is presented and we demonstrate the possibilities of content generation in a browser-based multi-player environment. In the sixth part, we discuss related work and finally, the seventh part concludes the chapter.

2. Web as a Gaming Platform

The architecture of a web application or web-based game often follows three-tier architecture (see Fig. 1) that is presented in more detail in [12]. The tiers are named as the presentation, business, and data logic. Traditionally, the browser has contained the presentation logic and acted as a client. The business logic runs on the web server and the data logic on the database server. However, a trend towards thicker clients is an emerging trend, and a number of advantages can be gained by moving some or the most of the business logic from server to browser.

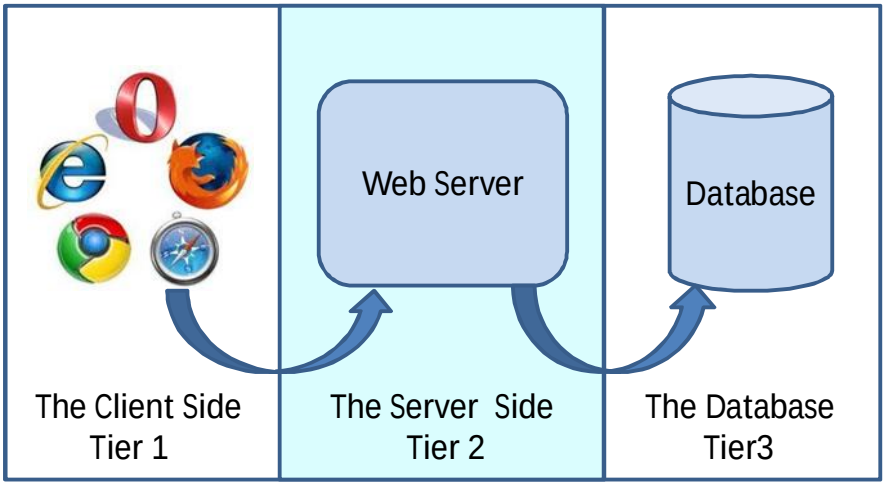

Figure 1. Three tier architecture of the web applications.

In designing browser-based game the partitioning between the client and the server is one of the key design problems. Two possible extremes are thin and thick client. In a thick client system, the functionalities and state of the game are located on the client side. In a thin client system, the game logic mostly runs on the server and the programming language for the game can be selected more freely and the source code is

not exposed to the players. Thin client approach makes cheating more difficult and prevents direct stealing of the source code. Furthermore, it is also possible that the game with a thin client works better in the mobile devices with limited processing capabilities. As a downside, the processing of each action requires a server request and the user experience depends on a lot of networking configuration. By lifting some or the most of the game logic from the server to the client it is possible to reduce communication and CPU needs and therefore serve more clients. For the user, the thick client provides a more responsive platform and often an improved Graphical User Interface (GUI). In some cases, a thick client can run totally independently after it has been downloaded from the server. Still, for the sake of persistence and communication, the thick client requires the server and at least a periodic internet connection.

The structure of the Web and somewhat outdated APIs of the browser have been limiting browser-based game development. Consequently, many more graphical games are implemented using plugin based techniques such as Adobe Flash [13]. However, the obstacles are being worked out and a trend away from the plugins seems to exist [14]. For example HTML5 [15] and WebGL [16] standards are offering an important set of tools for developing the client side and can be used through JavaScript APIs. HTML5 introduces new canvas element for drawing, methods for user interaction especially drag-and-drop support, offline web applications and audio support. Upcoming features that allow playing of audio and video in the browser without plug-ins can be very useful for game developers. WebGL utilizes a graphics processing unit (GPU) of the computer for drawing 2D and 3D graphics on a canvas. The hardware accelerated approach offers a long-awaited graphics boost for browser-based games and allows implementing complicated 3D games like Quake II for the browser [17].

The standards mentioned above strengthen the position of JavaScript among the game development language of the Web. Previously, performance issues have also been limiting the use of JavaScript when developing full-scale browser-based games. However, this obstacle is mostly withdrawn, since the performance of JavaScript engines in browsers has increased significantly in recent years.

3. Real-Time Multiplayer Games in Browser

Many installable games enable players to compete against or with each other over the internet. Typical examples are Call of Duty, Civilization, Counter Strike, and World of Warcraft, to name a few. The browser-based games have made their breakthrough in casual playing, while multiplayer games are not yet the main stream. The Web is strongly rooted in information sharing and the current technical solutions rather isolate users rather than make them aware of each other. Many problems exist when using the Web as a multiplayer gaming platform. The current technologies for creating browser-based applications are often referred to "Web 2.0" technologies. While Web 2.0 is mostly a marketing term, the applications still often users some interactions with other users [18]. As the users of the browser-based games are already connected through the game server, it seems as a natural addition to offer multiplayer games.

Still the majority of current browser-based games seem to be single player games or games that just indirectly enable the competition against other players. As an example, high score lists and Facebook posts generated by the game can be mentioned. Some games have asynchronous multiplayer features. FarmVille allows the player to perform various farm management related tasks such as planting, growing and harvesting crops. Players can also help friends in various farm related tasks, like watering the plants. In Urban Dead, the player controls a survivor or a zombie in the world after zombie apocalypse. The most common form of interaction seems to be punching or healing the characters that have run out of action points. We would like to see more games with real-time interaction and therefore in this chapter we introduce some problems and solutions related to real-time multi-user browser based games. In our chapter, the real-time does not implicate strict constrains on response time, but implies fast interactions between the users as defined in [19].

In the three-tier architecture of web application, the communication between the tiers is handled using asymmetric communication. Browser requests information from the web server, and the server makes queries for the database. As the server and database are not capable of initiating communication, the real-time communication between the players is difficult to initiate. In the simplest form, the problem can be evaded by making the client poll the updates. New requests are initiated and the updates that are stored in

the queue can be sent with responses. Depending on the need, polling can be done often or scarcely. Still, there always is a trade-off between the latency and the efficiency. If done too often, most of the responses do not contain any payload data, and bandwidth and processing power is constantly wasted; if done too scarcely, the updates have to queue before they get delivered and latencies grow. Several imitations of the server push exist. Most of them fall under an umbrella term Comet [20]. The downside of these approaches is that the server must keep connections open to all clients it updates and for avoiding the connection timeout, the connection has to be renewed by the time to time [21]. WebSockets [22] introduce a standard implementation for the server push and if they at some point come universally available among the browsers, the problem of bi-directional communication is solved.

Another free standardized open source API for real time communication is WebRTC that aims for real-time communication without plugins and is already available for Google Chrome [23]. WebRTC brings the peer to peer real-time communication (RTC) to browsers and thus fills one missing key piece of the platform. The attempt is part of the Chrome project, but it aims to cross industry solution. WebRTC offers MediaChannel and DataChannel for easily streaming media and data between JavaScript based client-side applications. The web server is still needed, but only for serving the content and for enabling users to discover each other. In gaming, the MediaChannel could be used for implementing communicative features like a point-to-point audio channel between the team members and the DataChannel could be used for streaming other game data. Compared with other solutions, like WebSockets, the big advantage of WebRTC is that the messages are traveling directly between the client applications and therefore the less computing power and bandwidth is needed on the server-side. The lag is also significantly reduced since as the messages need not to visit the server but can be routed directly using the shortest route between the clients.

WebSockets, Comet, and polling all enable bidirectional communication between the client and the server, and WebRTC between the two clients, but the cap between the server and database must still be crossed if we want to use a database as a messaging channel in a multiplayer game. The problem can be solved by utilizing database triggers, but using databases for real-time communication is generally difficult and increases lag.

In many of the games, some data must be permanently stored, but this data is often something that we do not need to mediate to other parties in real-time. For example, in two player Tetris where the players can see others actions on real-time, it is not reasonable to store game actions to the database, whereas high scores could be stored.

In many web application frameworks, the users seem to work in their own sandboxes that only interact within each other through the database. However, the direct communication between the users is not necessarily blocked and it may be easily enabled at the server. With a shared class or object it is easy to implement a capability for creating new message channels and enable the subscribing to channels. In publish/subscribe pattern (see Fig. 2) the users are subscribing and unsubscribing to specific communication channels based on the application logic and the interests. A publisher does not send the notifications directly to other users but publishes them for a channel that mediates the message [18].

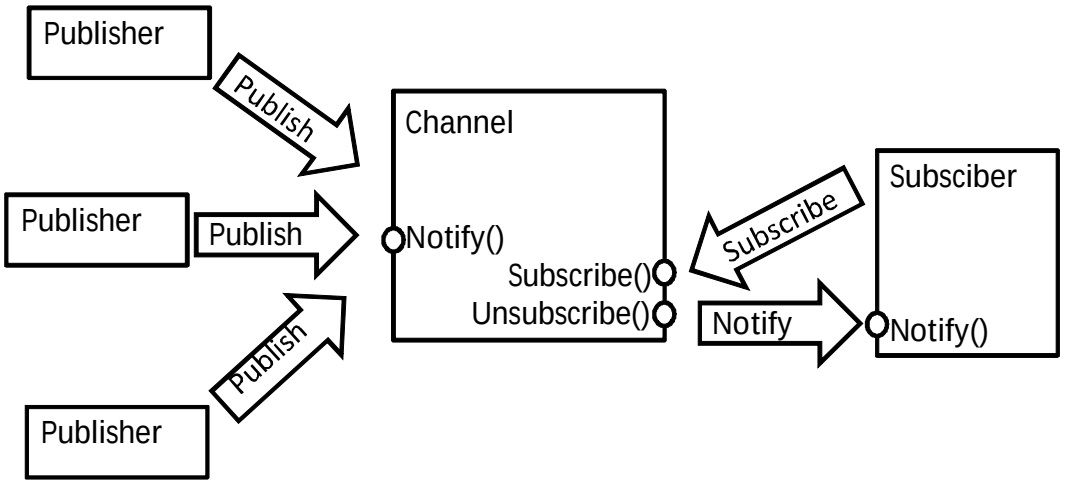

Figure 2. Publish/Subscribe pattern.

The sharing and synchronization of the data is another important aspect in multiplayer game. It is a matter of situation and opinion about how and where the data must be synchronized and what consistency model is to be used. Two extremes seem to be that the state of the game is stored at the server and mirrored to all clients. In this server driven alternative, WebSockets can be used for communication. In another extreme, where each of the client holds a unique perception to the state and they are

somewhat synchronized through the system, WebRTC could be used for enabling the communication between clients. In a first person shooter and other fast phased games, the estimation of the future data may be needed and later as the data is synchronized the estimation is adjusted to it.

In a game with large user base and lively data exchange, the data exchange and visibility between the users must be limited. This is needed for simplifying the problem and limiting the bandwidth usage. In most cases the interactions between the users are logically not from all-to-all, but more like from many-to-many. As a result, messaging that has no effect on a player can be eliminated. When limiting the visibility, the important concepts are: who can communicate to whom, who are visible to each other, and so on. In games, people in close proximity in the game world are visible to each other and can interact. For example, in old games like MUDs people in the same room can see each other, whereas in more recent game the game world can be more dynamically fragmented to areas in which players can interact and observe each other's.

4. Content Generation

Content generation can be used as a method to alleviating an enormous programming task of the game developers. Content is generated using a random or pseudo-random process. To be useful the generation process must be able to produce enough possible variation, but also be robust. Although, the content generation is an excellent method for reducing the burden of the application developers, it has some disadvantages as the randomly generated content does not easily relate to the game storyline [24]. Without additional operations it is possible, that game gets maybe gets unlimited amount of content, but the new content is not connected to the other parts of game. For example, rogue-like game Adom contains a pre-written narrative, but some of the dungeon levels are randomized because of replayability. For additional content, the game contains location called Infinite Dungeon that is not related to the story itself, but offers a player an infinite amount of dungeon levels for harvesting treasures and experience points.

The content to be generated depends on the type of the game. CRPGs can benefit from the generation of quests, non-player characters (NPCs), items, areas and computer

graphics. Content generation is especially good for CRPGs, because CRPGs typically contain huge fantasy worlds, which requires a lot of effort from the developers. The sizes of these fantasy worlds have been constantly growing as the genre has evolved. In addition, the sophisticated and long storylines of CRPGs are another good application area for content generation as the implementation of quests and storylines requires a massive amount of development work. For example, a part of the side-quests in CRPG can be generated instead of manual programming; therefore the number of side-quests and variation in the game can be increased without significant amount of extra work.

In some games, the random generation plays the very important role. In Slaves to Armok: God of Blood – Chapter II: Dwarf Fortress, for example, each game begins from generating the world completely from the scratch, with historical events and figures. The game is a good example of benefits of content generation, although it is using only ASCII graphics, the game has rich content and deep gameplay. The game is also very difficult and takes dozens of hours to learn to play it well. Minecraft, allows the players to build constructions out of textured cubes in 3d world. The world is mainly composed of 3d cubes that can be used for constructing buildings and items. In Minecraft, the gameworld is initially randomized Earth-like area and as the player moves towards the edges, the new areas are generated. It is said that it would be possible to stretch the size of generated gaming area to be nearly eight times than the surface of Earth before running into technical limits.

There are two different timing options when the content generation is executed. It can be done at 1) development time, before the distribution or 2) at run-time while the player is playing the game. Both of these have advantages and disadvantages.

4.1. Content Generation at Development Time

If the content is generated at development time, then the game developer explicitly starts the generation process and after the process is also able to verify the results. In case of non-satisfactory results, the process can be repeated as often as needed. Finally, when satisfactory content is generated, the game can be distributed to players. Later, when the game is at the hands of a consumer, it may be impossible to tell which parts of the game were programmed traditionally and which were produced by a content generation

algorithm. In games like Civilization, for example, both manually build and pre-generated verified maps can be included in the distributed package. Furthermore, if the player replays the game, the pre-generated content does not change for the new game. This feature allows experienced players to write more complete walkthroughs for the game to help beginners to solve the game.

The main disadvantage of a development time generation is that all the content is shipped with the game. This is problematic, because the size of the package grows. Especially this happens if the storyline of the game is heavily branched, for example because of many different races and character classes and there is different pre-generated content that varies based on these selections. By generating the content during the development time, the game also loses the ability to generate more material at the run-time and the replayability of the game does not differ from the game with manually created content. However, the pre-generated content is not as problematic for a browser based game as the content is stored on the server side and can therefore be altered on the fly. Furthermore, the whole package need not to be shipped because the client requests data based on need.

4.2. Content Generation at Run-Time

The content can also be generated at run-time while needed in the game. For example, in CRPG, the new background story can be presented to the player at each time when the new player character is created. The world can be created, for example by using fractal algorithms like in Armok. When descending to the dungeons, the new level or parts of it can actually be generated at the fly. While using content generation at run-time, in theory the game can offer an unlimited amount of content. Each time when the new game starts, the content could be new and fresh. In reality, the generation process is not able to generate an unlimited number of new interesting stories and areas and therefore it can drift to repetitious behavior if it is run enough times with starting values almost identical.

Disadvantages of run-time generation include the need for robust generation algorithms as the developer is not available to verify the results while the game is already running. Also, when using run-time generation, it is usually easier for the player to spot which parts of the game are generated, since those are different at each playtime.

However, this is a minor disadvantage. Traditionally, the content generation algorithms are shipped with the game and that sets some frames for the dispersion of generated content. However, in the games like Diablo III that requires constant internet connectivity, the content generation can be done on the server side and therefore algorithms can easily be extended later. As the browser-based games are constantly connected to the server, and therefore this same applies to them.

5. Example Game: Caves

To demonstrate the possibilities of content generation in a browser-based environment, we have implemented a browser-based prototype game. The game is implemented by using Node.js framework and therefore both; the client and the server side could be implemented by using JavaScript. The most client actions are resolved at the server (at tier 2 in the three lair architecture). However, certain data, for example the main attributes of player characters like experience points and cold pieces collected are written into the database for persistent storage. The usage of persistent storage allows the players to continue with the same characters at different playtimes. The used database used is MongoDB [25] that is an open source document-oriented NoSQL database. Since NoSQL database is used, we can store documents with dynamic schemas instead of SQL tables. MongoDB was chosen because of its easiness to use, certainly there is plethora of other options for a database to be used with server-side JavaScript.

The quests in the game are created at run-time based on quest templates. Each quest contains several rooms with monsters and to advance in the quest, the players have to defeat the enemies. A screenshot of Caves game in action is shown in Fig. 3. Two players are trying to defeat a group of orcs.

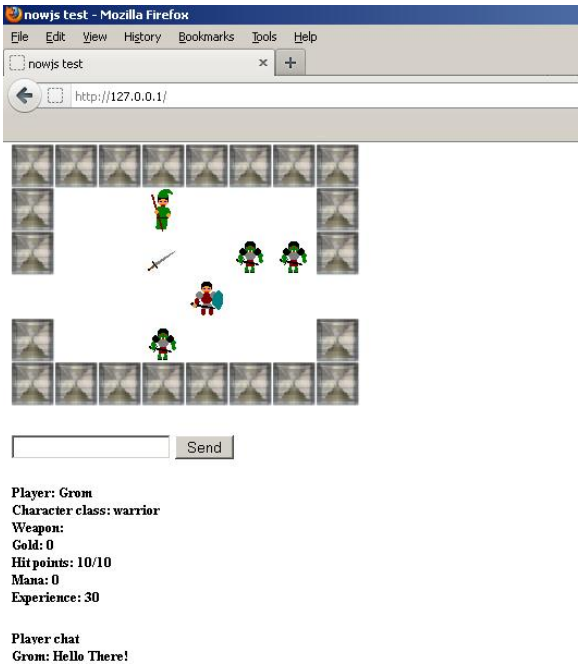

Figure 3. Caves game in action.

Caves game contains chat that enables players to communicate with each other while solving the quest although the fast processing gameplay of Caves limits messaging to only short messages. In Fig. 3, chat is visible below the player attributes. Another option for communication in a similar fast proceeding games is using an external program like Skype that allows players to talk each other using microphones, but that approach would compromise the anonymity of the players as the user identifiers used in another contexts should be shared. For allowing audio communication as a feature of the game, WebRTC could be used.

5.1. Game Architecture

For implementing Caves game Node.js that is a framework for writing highly scalable Internet applications was used. By using Node.js the whole web application is implemented with JavaScript. At the client-side the code runs on the web browser and at the server-side the code is run on V8 JavaScript engine that is an engine built for Google Chrome. Another software framework used in the implementation was NowJS, a framework built on top of Node.js. Fig. 4 presents the game architecture of Caves. The

overall game architecture follows the three tier architecture of web application presented earlier in Fig. 1.

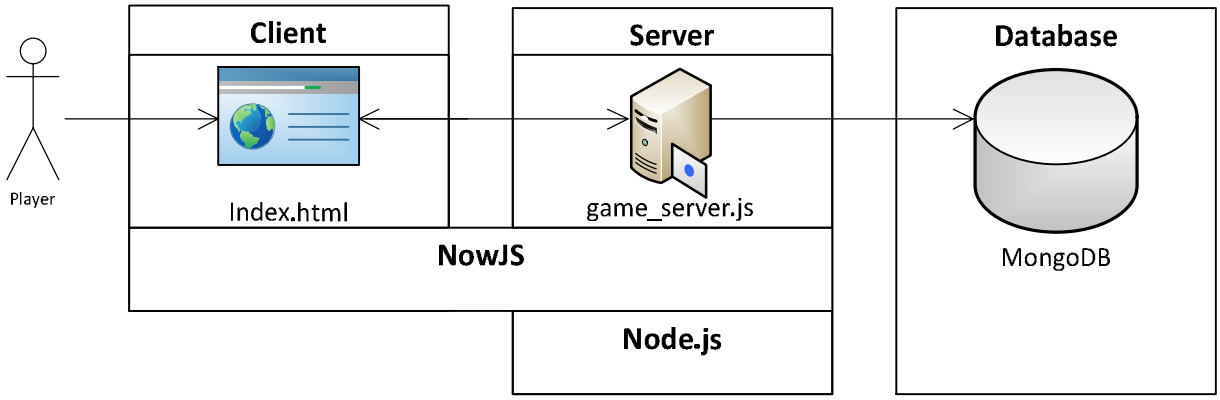

Figure 4. The game architecture.

The client-side is responsible for logging the players, and delivering player data to server. Similarly, when the game is running client parses moves of the player and delivers those to server, and results of those actions are resolved in server. After each quest the player data is written into the database.

The NowJS framework works on client and server, for application developer it makes those parts act like they were a single program. NowJS introduces a new magic namespace called now. The namespace `now` can be used for sharing synchronized variables between the client and the server and calling functions in server from client and vice versa. In Listing 1, we demonstrate the use of `now` namespace from the client-side. Listing shows the part of code, which resolves the results of the player actions. As the player character tries to move to a certain position, the client calls a server function `moveTo` to find out whether it is possible.

Listing 1. Function calling from the client.

```
1. if (player_moving) {
2.     now.moveTo(player, player_id, x, y);
3. }
4. updatePlayerStatsToUI(player);
```

In line 2, the client calls a server-side function and parameters are passed on the server to correspond the results. The `player` is an object that contains the attributes of the player character and `player_id` is a unique identifier assigned for the player. Coordinates `x` and `y` point out the location to where the player wants to move ones character. In line 4, the user interface of the client is synchronized for presenting the current situation. The `player_id` is not needed in the later function call, since in the client context in which it is performed; there is only one player character available.

Fig. 5 presents a sequence diagram of the same situation, a player making his move in the game. The player chooses a new direction for the character and presses a keyboard to change it. The player character is not moved instantly to the new location, instead only the moving direction is updated. The moving direction of the player character is all the time visible in the user interface. The player character is moved when the game loop commands all the player characters to move. This is done through the `now` namespace.

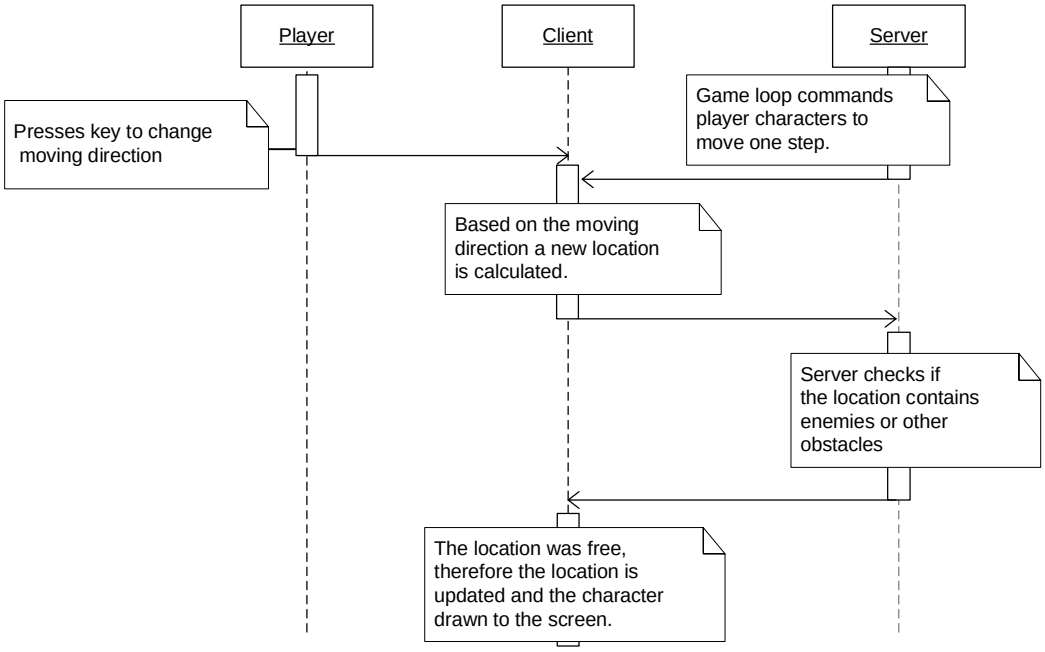

Figure 5. A player makes a move in Caves.

Similarly as player characters, the game monsters are moved. Currently monsters in level are moved towards the closest player character or to attack if they are already in adjacent location in the level.

5.2. Characters

In Caves, each player controls a fantasy character that starts at first experience level and collects experience points for further levels by adventuring and defeating enemies. Currently, there are two basic character classes for a player to select, but adding more character classes would be a simple task.

Our example character classes are Warrior and Wizard. Warriors are experts in close combat and dedicate their lives to practice this deadly art. They are also trained to use different weapons that can be found by searching the rooms. Wizards are weaker at close combat and they do not have as many hit points as warriors. Furthermore, they are not able to use weapons found in the rooms. However, they can blast enemies from a distance by using magical spells.

5.3. Quests

Quests in CRPGs are individual tasks for the player character to solve. Those have been one of the easiest ways for game designers to introduce storytelling elements into games [26]. Quests can also be used to guide the player through the game storyline. A common approach in CRPGs is that a game contains the main quest that is followed throughout the game and while advancing in the massive main quest, the smaller side quests are offered. By completing those smaller side quests, experience points and some rewards are gained. These can be experience points, money, items, meeting new NPCs or gaining valuable information about the game world.

In Caves, quests consist of series of rooms containing items and monsters. As the players have completed all the rooms of the quest, it is considered as solved and the players are able to save their characters and stop playing, or continue adventure by starting a new quest. The quests are generated at run-time by using quest templates. The quest templates are located in a separated directory on the server. When a new quest is

needed in the game, the server uses one of the templates and generates a new quest. The used template is chosen based on the experience level of the player character. Listing 2 shows the structure of a quest template configuration file.

Listing 2. An example quest template.

```
1. min_rooms: 2
2. max_rooms: 4
3. max_width: 9
4. max_height: 7
5. enemies: humanoids
6. material: stone
7. difficulty: 3
8. treasures: 24
9. treasure_size: 3
```

Lines one and two define low and high limits for the number of rooms in the quest. Respectively lines three and four define the maximum size of a single room. Enemies define the type of enemies that are used. At the moment, the only supported enemy types are humanoids (goblins, orcs and trolls), but new types would be straightforward to implement. Line six defines what material the room walls are made of. Currently, the possible material options are stone and wood. Line seven defines the base difficulty of the quest and therefore determines the number of enemies in the quest. Value three means normal difficulty, the range of difficulty values is from one to ten. Line eight tells the probability of a treasure in a room and line nine scales the size of the treasure. Again, three is normal and values can vary from one to ten.

5.4. Monsters

Rooms in quests are filled with monsters that must be defeated to complete the quest. In Caves there are three types of humanoid monsters:

Goblins are weak, but numerous. The rooms are inhabited by these evil, little creatures. They are usually destroyed by a single hit or a magical blast.

Orcs are basic enemies of Caves. They are stronger than goblins. However, they are still no match for well equipped, experienced warrior.

Trolls are strong and fearsome enemies, but luckily they are also rare. They are hard to defeat in combat, however they are still vulnerable to magic.

Listing 3 shows a piece of server-side code that determines how the monsters are created. As the room is generated, the `create_monsters` function is called and performed at the server. At this point, the details, i.e. how many rooms the quest will contain, are already known by the server. The function arguments are the number of monsters to be created and the name of monster type. The list named `monsters` contains of all currently active monsters, which are in the same room as the player characters. In the `create_monster` function attributes that are general to all monsters are initialized. Later rest of the attributes, for example how much experience is gained from killing the monster and hit points are initialized based on the type of the monster.

Listing 3. Monster creation.

```
1. function create_monsters(number, mon_name) {
2.    for (i = 0; i < number; ++i) {
3.       var mon = new Monster(mon_name);
4.       attackNearestPlayerTactic(mon);
5.       do {
6.          mon.x = random_number(level[0].length -1);
7.          mon.y = random_number(level.length - 1);
8.       }  while (!isFree(mon.x, mon.y));
9.       monsters.push(mon);
10.    }
11. }
```

In line 3 a new monster is created and in line 4, the created monster is initialized to move towards the nearest player, and attack if they are in the adjacent square. Although the solution is simple, it is reasonably fair and predictable for the players and suits well into multiplayer games containing large enemy swarms. The same solution has been used in a legendary arcade game Gauntlet and its later versions. Most likely a more complex solution would be needed if monsters with advanced behavior were added into

the game. By advanced behavior we mean e.g. monsters that could attack from a range instead of close combat. Inside do while –loop, lines 6 and 7 the starting location for monster is given. A new place is drawn until it a free one is found. The variable level is a two dimensional array containing the current room. When the monster has been created, it is pushed into an array containing monsters of the current room in line 9.

5.5. Lessons Learned from Proof-of-Concept Implementation

The use of software framework Node.js and its extension NowJS speeds up the game development significantly. Some kind of synchronization between the server and clients is mandatory while implementing multiplayer browser-based game and therefore it is nice to have the framework that offers the easy synchronization of variables and functions as a feature. To prevent cheating, the content generation algorithms and template files have to be stored and executed at the server-side and only results are exposed to the clients. By using modern browser plugins, it is easy to investigate and modify the JavaScript code and therefore the visibility of program code should be limited whenever possible so that the players cannot gain unfair advantage. Cheating has always been problem in online multiplayer games, whether those have been implemented using installable client software or not, therefore it should be prepared already in the development phase of the game. Browser-based implementation opens new ways of cheating like browsing the source code for advantage, since it is usually more easily accessible in browser-based environment than it was in compiled binary format.

As overall, JavaScript language with described frameworks suited well for developing games to browser-based environment. However, debugging applications to find bugs can sometimes be difficult since JavaScript is not compiled, but interpreted on the fly by the JavaScript engine of the browser. Furthermore, JavaScript is an error tolerant until the end and often the error that causes the problem has happened a long before it causes actual effects. For example, if the developer makes a typo while trying to assign a new value for variable, no error happens, just a new variable with the typo is introduced and a value is assigned for it. Errors of that kind can be very difficult to pinpoint. Some tools like FireBug and Google Chrome Developer Tools exists, but most often more traditional ways like printing test prints and outputting client messages to the

user interface has be used to find bugs. A good solution for implementing a JavaScript application is to create the application incrementally and test new and old features as often as possible. Also the multiplayer features should be tested in early phase and tests should be conducted through the whole development process.

In the following we present the rights and wrongs of proof-of-concept implementation.

The rights:

1) NowJS enables sharing variables and function calling between client and server, which rapidly speed up the development.
2) Modular architecture enables extending the game easily. For example adding new character classes, monsters and quest templates is very easy.
3) The implementation is very light-weight and a regular laptop computer can be used for running server of the application without performance problems.

The wrongs:

1) Testing of JavaScript applications can be difficult, since there is no error messages as the application is not compiled. This can be addressed by iterative development process.
2) The updating of user interface is not as smooth as it we wanted, and some jitter can be noticed when playing the game. However, similar synchronization problems were found on other applications using the same frameworks, described in the related work section.

6. Related Work

Currently there are a couple of tools for quest generation, although those are used in traditional binary environments. GrailGM [27] is a run-time game master that offers quests and actions to the player. GrailGM acts as a game master and finds suitable quests from the quest library for the player character at the run-time. The offered quests are based on the history of the player character and current world state. Traditionally, CRPGs

relay on combat-based quests, since the combat systems of CRPGs have been robust for a long time and those are easier to implement. Combat-based quests are also an easier target for content generation tools. However, GrailGM aims at creating non-combat quests. Furthermore, the special abilities of the player character can also be used in the quest selection process, and those can be used in the plot of the selected quest.

SQUIGE [28] is a side-quest generator designed for automatically producing side-quests. NPCs and other elements for a side-quest are given to SQUIGE and it produces quest outline based on those. Afterwards the developer reviews the quests and accepts or declines (and asks the system to produce another) the outline. The developer also adds dialogue and places the items for the quest. Finally, the quest is given to a programmer or a scripting system, which creates the script to control the side-quest.

A few game examples using the same frameworks as us can be found on the Web. Although none of those seem to use content generation. Browser-based games [29] presents a good tutorial on how to implement multi-player browser based games using NodeJS and NowJS. The games and applications utilizing NodeJS and NowJS most often offer multi-user real-time features and therefore it is easy to find examples just by using search engines. For example, Browserquest [30] is a MMORPG where players are adventuring together and the content generation methods presented in this chapter could be easily applied to the game. Another example, Fireflies [31] is a collaboratively jumping puzzle. Furthermore, NowJS has organized a hackathon for implementing real-time multiplayer games and the results like Multiplayer Tetris and Battle snakes are available for playing in here: http://nowjs.com/rtt/1/results. By studying the examples we come to a conclusion that games with fast real-time interactions and competitive aspect like Battle Snake are problematic, as the smallest synchronization problems or lag may evoke the feeling that game is not fair. The competitive games with less real-time interactions between the players like Multiplayer Tetris seem to be a more viable alternative. Collaborative games like Browserquest and Fireflies where players have the shared goal are easier approach since the small problems can be solved just by trying again and it does not matter so much which of the players achieved the goal. As a side note, in Browserquest, the online communication between the players was enabled, but it

would be beneficial for other examples too, although the missing communication can also be seen as a game element.

Ben Nadel has written a blog article [32] about Realtime Messaging And Synchronization With NowJS and Node.js and discusses about how the shared scope between the local and remote context blurs the line between client and server code and thus makes the synchronization of states easier. Furthermore, he discusses on how the capability to invoke client-based functions from the server and visa verse makes the life of a developer even easier. These are the same conclusions that we made after our prototype game project.

7. Conclusions

The current paradigm shift towards web-based software is visible among all kinds of software and there is no reason to assume that gaming could be some sort of an exception. The number of browser-based games has exploded in recent years and technologies for implementing them are evolving at fast rate. Although use of content generation is still rare in browser-based games, it seems to offer as huge potential in browser-based environment as it offers in binary environment. By using content generation, new additional content can be generated instantly as it is needed in the game. When playing the browser-based games, the users are connected through the server, and therefore it seems only reasonable to assume that the games are not played alone but together. Although most of the current browser based games are still single player games or games where competing happens asynchronously through highscore lists, the number of multiplayer browser-based games has increased rabidly in recent years and we believe that this development accelerates as frameworks and tools used to developed browser-based games mature.

In this chapter, we discussed possibilities and limitations of content generation in browser-based multiplayer environment. To demonstrate these possibilities, we have implemented a proof-of-concept browser-based game called Caves. Although Caves is a simple example, it shows true our assumptions that run-time content generation can be used in browser-based game environment and that implementation of multi-player games

is possible even with the frameworks of today. Another studied concept was how to use JavaScript language in both client and server-side. By utilizing JavaScript and frameworks, it was possible to implement Caves with a rather small code base. The software frameworks we used as a base of implementation were Node.js and its extension NowJS. Both of the frameworks carried out well what they promised and speeded the game development significantly. Based on the experiences collected while implementing the game, it seems that JavaScript environment scales up also for more complex games.

References

[1] A. Taivalsaari, T. Mikkonen, M. Anttonen and A. Salminen, “The Death of Binary Software: End User Software Moves to the Web," in *Proc. 9th International Conf. on Creating, Connecting and Collaborating through Computing (C5),* Kyoto, Japan, 2011, pp.17-23.

[2] M. Jazayeri, "Some Trends in Web Application Development," in *Proc. Conf. on Future of Software Engineering (FOSE),* Minneapolis, MN, 2007, pp.199-213.

[3] J-M. Vanhatupa, “On the Development of Browser Games - Technologies of an Emerging Genre,” in *Proc. 7th International Conf. on Next Generation Web Services Practices (NWeSP),* Salamanca, Spain, 2011, pp. 363-368.

[4] Google Web Toolkit Overview. Available: https://developers.google.com/web-toolkit/overview

[5] M. Grönroos, *Book of Vaadin*. Uniprint, 2012.

[6] Node.js platform, Available: http://nodejs.org/

[7] J-M. Vanhatupa, “Browser Games for Online Communities,” *International Journal of Wireless & Mobile Networks (IJWMN),* vol. 2, no. 3, pp. 39-47, Aug. 2010.

[8] Facebook social utility, Available: http://www.facebook.com/

[9] D. Anderson, “The Dark Side MMOGs: Why People Quit Playing,” in *Proc. 14th Conf. on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGames),* Louisville, Kentucky, 2009, pp. 76-80.

[10] J. Smed and H. Hakonen, *Algorithms and Networking for Computer Games,* John Wiley & Sons Ltd., West Sussex, England, 2006.

[11] NowJS library, available: http://nowjs.com/

[12] R. Peacock, "Distributed Architecture Technologies," in *IT Professional*, vol. 2, no. 3, pp. 58-60, May/Jun 2000.

[13] Adobe Flash Platform. Available: http://www.adobe.com/flashplatform/

[14] A. Taivalsaari, T. Mikkonen, D. Ingalss and K. Palacz, "Web Browser as an Application Platform: The Lively Kernel Experience," Sun Microsystems Laboratories Technical Report TR-2008-175, Jan. 2008.

[15] I. Hickson and D. Hyatt (editors). 2010. *HTML5. W3C Working Draft,* Available: http://www.w3.org/TR/2010/WD-html5-20100304/

[16] Khronos Group. 2011. *WebGL Specification, Version 1.0. Technical Specification*, Available: https://www.khronos.org/registry/webgl/specs/1.0/

[17] M. Anttonen, A. Salminen, T. Mikkonen and A. Taivalsaari, "Transforming the Web into a Real Application Platform: New Technologies, Emerging Trends and Missing Pieces," in *Proc. the 26th ACM Symposium on Applied Computing, TaiChung, Taiwan, (SAC),* 2011. pp. 800-807.

[18] P. Eugster, P. Felber, R. Guerraoui and A-M. Kermarrec, "The Many Faces of Publish/Subscribe," in *ACM Computing Surveys*, vol. 35, no. 2, pp. 114-131, Jun. 2003.

[19] C. Ellis and S. Gibbs, "Concurrency Control in Groupware Systems," in *Proc. the 1989 ACM SIGMOD international conference on Management of data*, New York, (SIGMOD'89), 1989, pp. 399-407.

[20] A. Russell. 2006. *Comet: Low Latency Data for the Browser*, Available: http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/

[21] J. Resig, *Pro JavaScript TM Techniques,* Springer-Verlag, New York, 2006, pp. 287-304.

[22] I. Hickson, *The Web Sockets API*, Available: http://www.w3.org/TR/websockets/

[23] A. Bergkvist, D. Burnett, C. Jennings and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers," Available: http://dev.w3.org/2011/webrtc/editor/webrtc.html

[24] J-M. Vanhatupa. “Towards Extensible and Personalized Computer Role-Playing Game Fantasy Worlds,” in *Proc. 4th Computer Science and Electric Engineering Conference (CEEC)*, Colchester, UK, 2012, pp. 187-190.

[25] MongoDB an open source NoSQL database, Available: http://www.mongodb.org/

[26] S. Tosca, “The Quest Problem in Computer Games,” in *Proc. First International Conf. on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE 2003)*, Darmstadt, Germany, 2003.

[27] A. Sullivan, M. Mateas and N. Wardrip-Fruin, ”Rules of Engagement: Moving Beyond Combat-based Quests,“ in *Proc. Intelligence Narrative Technologies III Workshop*, ACM, New York, 2010.

[28] C. Onuczko, D. A. Szafron and J. Schaeffer, “Stop Getting Side-Tracked by Side-Quests,” in *AI Game Programming Wisdom 4*. Charles River Media, 2008, pp. 513-528.

[29] Building a Multiplayer Game Server, available: https://sites.google.com/site/vadimtutorials/assignments/9buildingamultiplayergameserver

[30] Browserquest game, http://browserquest.mozilla.org/

[31] Fireflies, game, http://bearhanded.com/fireflies-our-html5-multiplayer-game/

[32] B. Nadel, “Realtime Messanging and Syncronization with NowJS and Node.js, available: http://www.bennadel.com/blog/2171-Realtime-Messaging-And-Synchronization-With-NowJS-And-Node-js.htm

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3170-5
ISSN 1459-2045